

# EXPLORING DISCRETE DYNAMICS SECOND EDITION

---

---

*The DDLab Manual*

Tools for researching Cellular Automata,  
Random Boolean and Multi-Value Networks, and beyond.

## ABOUT THE AUTHOR

Dr. Andy Wuensche is a former architect, the inventor of reverse algorithms for cellular automata and discrete dynamical networks, the author of many academic publications including the book *The Global Dynamics of Cellular Automata* (with Mike Lesser) in the Santa Fe Institute's *Studies in the Sciences of Complexity*. He created and continues to develop the classic software *Discrete Dynamics Laboratory (DDLab)*, widely used in research and education. He has lectured at universities and conferences throughout the world. He is an independent academic, a visiting Professor at the International Center of Unconventional Computing, UWE, Bristol, and a visiting research fellow at the Dept. of Informatics, University of Sussex.



Oil on canvas by Paul Wuensche  
<http://paulwuensche.com/>

ANDREW WUENSCHÉ

---

---

EXPLORING DISCRETE DYNAMICS  
SECOND EDITION

---

---

*The DDLab Manual*

Tools for researching Cellular Automata,  
Random Boolean and Multi-Value Networks, and beyond.



Luniver Press  
2016

Published by Luniver Press  
Frome BA11 3EY United Kingdom

British Library Cataloguing-in-Publication Data  
A catalogue record for this book is available from the British Library

EXPLORING DISCRETE DYNAMICS Second Edition  
(First Edition 2011 Luniver Press)

Copyright © Luniver Press 2016

The cover images were created with DDLab. A space-time pattern ( $n=600$ ) and a basin of attraction ( $n=15$ ) are illustrated — both according to the same  $v2k5$  one-dimensional cellular automata rule (hex) 3dae2997.

All rights reserved. This book, or parts thereof, may not be reproduced in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage and retrieval system, without permission in writing from the copyright holder.

ISBN-10: 1-905986-47-5  
ISBN-13: 978-1-905986-47-7

While every attempt is made to ensure that the information in this publication is correct, no liability can be accepted by the authors or publishers for loss, damage or injury caused by any errors in, or omission from, the information given.



*To Stephanie*

# Chapters

	Preface .....	vii
	Contents .....	xiv
	List of figures .....	xxxix
	List of tables .....	xxxviii
1	Overview .....	2
2	Summary of DDLab functions .....	11
3	Accessing DDLab .....	28
4	Quick Start Examples .....	34
5	Starting DDLab .....	56
6	The first prompt in DDLab .....	65
7	Value-range, $v$ .....	71
8	1d network size $n$ , or range- $n$ .....	75
9	Neighborhood, $k$ , or mixed- $k$ .....	83
10	The local neighborhood, and network geometry .....	90
11	Setting the wiring, quick settings .....	97
12	Setting special wiring .....	101
13	Rules types .....	110
14	Rulemix options .....	121
15	Setting Canalization in a random rulemix .....	135
16	Setting a single rule .....	142
17	Reviewing network architecture .....	169
18	Transforming network rules .....	202
19	File/print network architecture .....	210
20	The network-graph, and attractor jump-graph .....	221
21	The Seed or initial state .....	244
22	The Derrida plot .....	271
23	Graphic conventions for attractor basins .....	280
24	Output parameters for attractor basins .....	286
25	Layout of attractor basins .....	309
26	Display of attractor basins .....	318
27	Pausing attractor basins, and data .....	331
28	Mutation of attractor basins .....	347
29	Final options for attractor basins .....	355
30	Drawing attractor basins, and changes on-the-fly ...	378
31	Output parameters for space-time patterns .....	386
32	Drawing space-time patterns, and changes on-the-fly	422
33	Classifying rule space .....	475
34	Learning, forgetting, and highlighting .....	497
35	Filing .....	511
36	Vector PostScript capture of DDLab output .....	517
37	Glossary .....	519
	References .....	525
	Index .....	529

# Preface

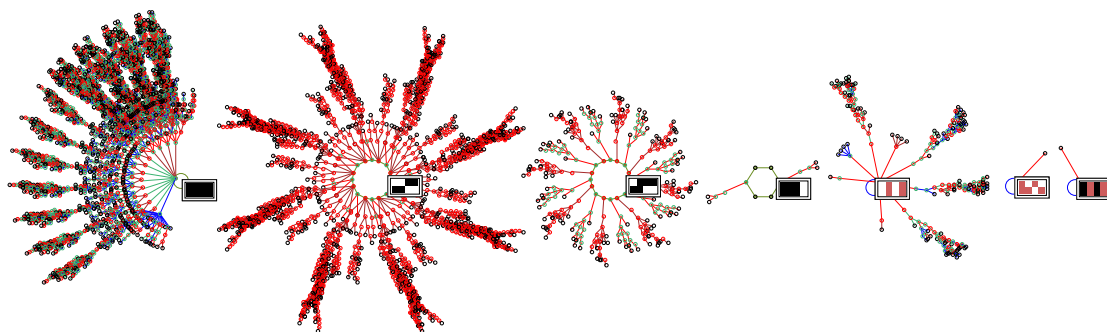


Figure 1: The basin of attraction field of a 3-value Cellular Automaton,  $v=3$ ,  $k=3$ ,  $n=8$ . There are in fact 17 basins, but equivalent basins have been suppressed leaving just the 7 prototypes. One attractor state is shown for each basin. The rcode rule-table is 020120122021201201011011022.

---

## Dynamics *on* networks, and *of* networks

Networks of sparsely inter-connected elements with discrete values 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 science and economics; to parallel computation, emergence, self-organization and artificial life; to complex systems of all kinds.

Dynamics *upon* or *on* a network are its changing patterns of cell-states, while the underlying network architecture, its connections and logic, remains fixed. DDLab studies these changing patterns and how they “fall” along trajectories into 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 plasticity, 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 its state 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 possibility of all sorts of “emergent” dynamic patterns.

We can unravel the dynamics and gain some insight into the basic principles by studying idealized networks where 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 [31, 29, 30], collision-based computing [1], neural networks, memory and learning [34], genetic regulatory networks [20, 25, 16, 33, 36], 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 utilizes interactive graphics and other methods to study cellular automata (CA), random Boolean networks (RBN) [20], 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 behavior to be explored — mostly terra incognita.

There are currently compiled versions<sup>1</sup> of DDLab for Linux, Mac, Cygwin and DOS. The code is written in C, and is open source “free software” under the GNU General Public License. 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 [31], analogous to Poincaré’s “phase portrait” which provided powerful insights in continuous dynamical systems. A key insight is that the dynamics on networks converge, thus fall into a number of basins of attraction. The state transition graphs, consisting of trees rooted on attractor cycles, represent a network’s memory, its ability to hierarchically categorize its patterns of activation (state-space) as a function of the precise network architecture [33].

Relating this to space-time patterns in CA, high convergence implies order, low convergence implies disorder or chaos [31]. The most interesting emergent structures occur at the transition, sometimes called the “edge of chaos” [22, 38].

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. Most images can be captured as vector PostScript files suitable for publication. This book provides a comprehensive operating manual for the current multi-value version of DDLab at the time of publication.

<sup>1</sup>Earlier compiled versions are also available for Unix and Irix.

---

## First edition updates

The items below were the significant DDLab updates (since the 2001 online manual) documented in the the 2011 first edition of Exploring Discrete Dynamics [50].

### multi-value

DDLab was generalized for multi-value logic, with up to 8 values (or cell-states, or colors), instead of just Boolean logic (two values: 0,1). Of course, with just 2 values selected, DDLab behaved as before. The multi-value version opened up new possibilities for dynamical behavior and modelling.

### totalistic rules

DDLab could be constrained 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 option reduced the relative sizes of rule-tables allowing larger neighborhoods, with an upper limit of 25 instead of 13, at the cost of disabling basin of attraction functions. In 2d the neighborhoods were predefined to make hexagonal as well square lattices. Many interesting cellular automaton rules with “Life-like” and other complex dynamics were found in totalistic multi-value rule-space, in 3d as well as 2d [44, 45].

### vector graphics

Options were provided to capture most of DDLab’s graphic output, including space-time patterns and attractor basins, the network-graph and the attractor jump-graph, as vector PostScript files, which is a preferred format for publication than the previously available bitmap images.

---

## Second edition updates

A number of significant DDLab updates have been released since the first edition of Exploring Discrete Dynamics was published in 2011, which are now included and documented in this second edition, as well as various other improvements and corrections. The significant updates, listed in no particular order, are as follows:

### greater system sizes

DDLab now allows greater sizes in a range of parameters (summerised in section 1.6) — value-range, neighborhoods and rule-tables (chapter 7), and network size (sections 1.6.1, 8.3). These greater sizes can be extended even further with a new initial “exLimits” option (section 2.2.3). Greater sizes of networks are especially useful for 2d and 3d space-time patterns, but this presented a challenge in displaying, manipulating and filing which have been resolved. Maximum sizes depend on your CPU and DDLab version (32-bit or 64-bit), and available RAM.

## network “block” in 2d and 3d

As a consequence of larger network sizes, in 2d and 3d the new default when defining a network block (its wiring and/or rules) is to show just the block edges, to speed up the graphics presentation for large blocks, but this can be toggled to show the block in full as before (sections 17.7.5 and 17.8.5). When a block is active, setting random or special wiring applies either within the block (as before), or (a new option) to the remainder of the network outside the block.

## use of the mouse pointer in the wiring graphic

New functions in the wiring graphic (section 17.5) allow a mouse pointer/click to reposition the active cell, and the last two clicks to define the default corners of a block. This makes it easier to explore and amend network wiring. The pointer/click methods apply to any wiring graphic, 1d, 1d circle layout, 1d as 2d, 2d (square or hex), and 3d where the pointer moves on the 2d version of the 3d network.

## neighborhood size extended up to 27 in all dimensions

The maximum neighborhood size  $k$  is now extended up to 27 cells, and predefined for 1d, 2d (square or hex) and 3d. New and modified 3d neighborhood templates all now fit within a  $3 \times 3 \times 3$  volume (figure 10.3).

## 2d hex/triangular neighborhoods

New 2d hex/triangular neighborhoods sizes  $k=3$  and  $k=4$  are introduced (figure 10.2) which permit investigating the dynamics on these simpler lattices, with many instances of complexity, for example figure 2.2.

## null boundary conditions

All DDLab functions and options can now be applied to systems with null boundary conditions (NBC), whereas previously only periodic boundary conditions (PBC) were available. This includes multi-value, CA in various dimensions, but also RBN and DDN (section 2.7).

NBC are of interest in pattern recognition, and applications where the system is grounded or quenched, or bounded by an edge, skin or membrane. As for PBC, NBC are also interesting as mathematical/dynamical systems in their own right.

## saving/loading a seed as an ASCII string and in Golly format

As well as the standard binary DDLab format, a seed can now be saved/loaded as an ASCII string (section 21.10) which is useful for interchanging the seed between DDLab and alternative software. A 2d seed can also be saved/loaded as an ASCII string following the “Golly” file format (section 21.11), software used in the game-of-Life community.

## saving the rule as a 1d seed file

The rule-table pattern can be saved as a 1d seed file (section 16.4.4), allowing a rule string to seed a basin of attraction, which itself is able to classifying rule space [6, 7].

## repetitive blocks of rules

When setting a rulemix from a limited subset of rules (section 14.4.2) as well as assigning rules randomly from the subset, the subset itself can be assigned to form repetitive blocks of rules — thus implement so called “hybrid CA” or HCA.

## transient data

In the “attractor histogram” (section 31.7) which provides statistical attractor data by running forward, a new option allows data to be recorded on the set of unique transients found so far, to each attractor, without duplication (figure 31.19). This provides additional data required for models of genetic regulatory networks.

## dynamic trails for gliders

Mostly effective in 2D CA to highlight mobile configuration such as gliders, glider-collisions, and glider-guns, a new feature allows these objects to leave green trails of specified length (section 32.11.1, figure 32.21).

## lattice and rule presentation

Improvements have been made to the graphic presentation of both states (seeds) and rules, including the color of the division lines of the current lattice (figures 21.7, 32.14).

## manipulating a predefined 2d patch

All preexisting options for manipulating and editing 2d states (seeds) can now be applied to just a predefined patch set by the last 2 mouse clicks. This is useful to design, edit, manipulate and duplicate configurations for 2d complex rules, such as the game-of-Life [10], the Spiral rule [45], and the X-rule [13]. The options include: jump, copy, move, spin, flip, compliment, and save the pattern or patch as a DDLab seed or vector PostScript file (figure 21.6).

## equivalent CA and the Derrida plot

New Derrida plot options include automatic plots of equivalent CA (figure 22.4), and lists of equivalence classes and rule clusters [31] (section 22.7).

## display of the bare attractor for very large networks

For single basins, just the bare attractor can be displayed by setting the number of backward steps to zero (section 29.5.1) so that reverse algorithms do not come into play. This new function allows the graphic image of the attractor for very large networks, including 2d, to be generated as in figures 29.2 and 29.3.

## reaction-diffusion dynamics in 3d as well as 2d

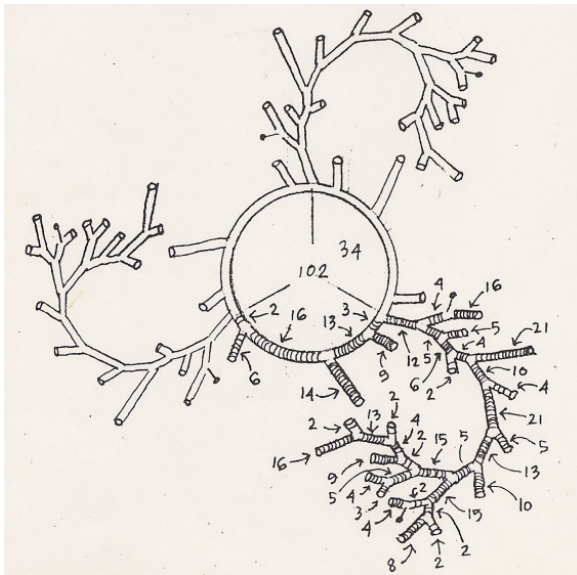
Reaction-diffusion dynamics (section 13.8) can be applied in 3d networks (figure 13.4) as well as in 2d as before (figure 13.3).

## return map by density

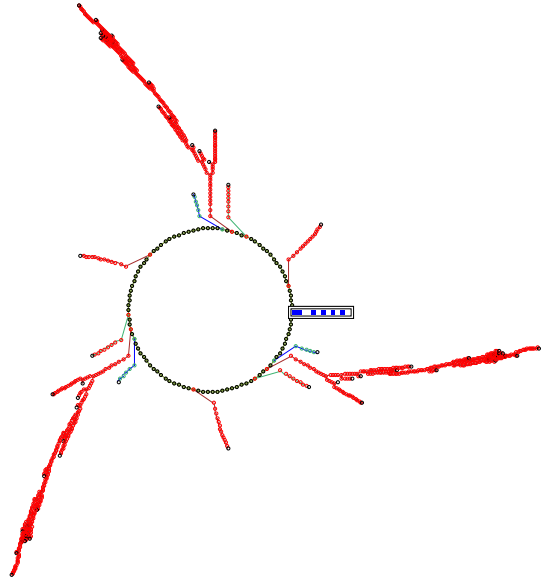
A scatter plot of the return map by density can be created, plotting the density of each value at  $t_{-1}$   $y$ -axis, against  $t_0$   $x$ -axis (section 32.12.8) as in figure 32.33. This can be interesting when the densities pulsate following a periodic oscillation, which occurs for a network with random wiring but one complex glider rule, for example rules from the the complex rule collection in section 3.6.2.

## resetting transient parameters

The transient scale can be reset, independently from the basin scale and attractor radius (section 25.2.4). When drawing transients, the gap between successive transient levels away from the attractor or subtree root is made to decrease asymptotically according to default parameters — applied for drawing attractor basins (state transition graphs) as illustrated throughout this book. However, new options allow these transient parameters to be varied for greater control of the presentation (section 25.2.2). This is especially useful to curtail the projection of very long transients, characteristic of chaos.



a pencil drawing from the very early days before automatic computer drawing was perfected



the same basin of attraction drawn by the graphics algorithm — transients curtailed (scale=15 F=6)

Figure 2: A single basin of attraction, elementary rule 30 — a chaotic rule,  $n=12$ , attractor period=102, maximum transient levels=126 — further examples in figure 25.4.



---

## Acknowledgements

The precursor of DDLab was the *Atlas* software, included on diskette inside the back cover of *The Global Dynamics of Cellular Automata, An Atlas of Basin of Attraction Fields of One-Dimensional Cellular Automata* [31], by Andrew Wuensche and Mike Lesser, published in 1992 in the Santa Fe Institute's *Studies in the Sciences of Complexity*. The *Atlas* software was compiled in DOS, with the operating instructions in Appendix 1 of the book. The book and original software can be found at <http://www.ddlab.org>.

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, Genaro Martnez, 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.

---

# Contents

<b>Chapters</b>	vi
<b>Preface</b>	vii
<b>Contents</b>	xiv
<b>List of figures</b>	xxxix
<b>List of tables</b>	xxxviii
<b>1 Overview</b>	<b>2</b>
1.1 Introduction . . . . .	2
1.2 Source code and platforms . . . . .	3
1.3 Discrete dynamical networks . . . . .	4
1.4 Space-time patterns and attractor basins . . . . .	6
1.4.1 Totalistic rules - forwards-only . . . . .	8
1.5 Categorization . . . . .	8
1.6 Size limits: network $n$ and neighborhood $k$ . . . . .	8
1.6.1 Network size limits . . . . .	8
1.6.2 Neighborhood size limits . . . . .	9
1.7 Parameters and options . . . . .	9
1.8 Measures and data . . . . .	9
1.9 Contents summary . . . . .	9
<b>2 Summary of DDLab functions</b>	<b>11</b>
2.1 DDLab's prompts . . . . .	11
2.2 Initial choices . . . . .	11
2.2.1 Totalistic rules, forwards-only, TFO-mode . . . . .	11
2.2.2 Basin field or initial state . . . . .	11
2.2.3 Exceeding normal limits of $n$ and $k$ . . . . .	12
2.3 Setting the value-range . . . . .	12
2.4 Setting the network size . . . . .	12
2.5 The neighborhood $k$ or $k$ -mix . . . . .	12
2.6 Wiring . . . . .	13
2.7 Null boundary conditions . . . . .	14
2.8 Rules . . . . .	14
2.9 Initial network state, seed . . . . .	15
2.10 Networks of sub-networks . . . . .	16
2.11 Presentation options . . . . .	16
2.11.1 Space-time patterns . . . . .	16

2.11.2	Attractor basins	17
2.11.3	Interrupting a run	18
2.12	Graphics	18
2.13	Filing and Printing	19
2.13.1	Filing network parameters	19
2.13.2	Filing data	19
2.14	Vector PostScript images	20
2.15	Bitmap images	20
2.16	Printing the screen image	20
2.17	Mutations	20
2.17.1	Running Forward	20
2.17.2	Running Backward	21
2.18	Quantitative, statistical and analytical measures	21
2.18.1	Behavior parameters	21
2.18.2	Network connectivity	21
2.18.3	Measures on local dynamics	22
2.18.4	Measures on global dynamics	23
2.19	Reverse algorithms	24
2.19.1	1d CA wiring reverse algorithm	25
2.19.2	Nonlocal wiring algorithm	26
2.19.3	Exhaustive reverse algorithm	26
2.20	Random map	26
2.21	Asynchronous and Sequential updating	26
2.21.1	Neutral order components	27
2.22	Sculpting attractor basins	27
<b>3</b>	<b>Accessing and running DDLab</b>	<b>28</b>
3.1	The DDLab web site	28
3.2	DDLab at SourceForge	28
3.3	Unzipping and running — Linux-like versions	28
3.3.1	Unix library files	29
3.4	Unzipping and running - DOS	29
3.5	The Quick Start Examples	30
3.6	Extra data files	30
3.6.1	Complex rule collections	30
3.6.2	Selected 2d complex rules, $v=3$	30
3.6.3	Selected seeds	31
3.6.4	Sorted rule samples	32
3.6.5	Byl's self reproducing loop	32
3.7	Copyright, License and Registration	33
3.8	Source code, and compiling	33
3.9	Previous versions of DDLab	33
<b>4</b>	<b>Quick Start Examples</b>	<b>34</b>
4.1	The DDLab screen	34
4.1.1	User Input	34
4.1.2	Backtrack	35
4.1.3	Quitting DDLab	35
4.1.4	Skipping Forward	35
4.1.5	The graphics setup	35
4.2	Basin of attraction fields	35

4.2.1	Changing basin parameters	36
4.3	Backwards space-time patterns, and state-space matrix	37
4.4	Basin of attraction fields for a range of network sizes	38
4.5	A single basin of attraction	39
4.6	A subtree	40
4.7	Space-time patterns — SEED-mode or TFO-mode	41
4.8	1d Space-time patterns	41
4.8.1	1d ring of cells, and scrolling the ring	43
4.8.2	Scrolling the ring	43
4.8.3	Multi-value 1d space-time patterns in TFO-mode	44
4.8.4	Noisy space-time patterns	45
4.9	2d Space-time patterns	46
4.9.1	2d space-time patterns — game-of-Life	47
4.9.2	2d Space-time patterns — binary totalistic rules	49
4.9.3	Multi-value 2d space-time patterns in TFO-mode	50
4.10	3d Space-time patterns	51
4.10.1	3d Space-time patterns — examples	51
4.11	“Screen-saver” demo	52
4.12	RBN and DDN	53
<b>5</b>	<b>Starting DDLab</b>	<b>56</b>
5.1	Running in Linux-like operating systems	56
5.1.1	Linux within Windows — Cygwin	57
5.1.2	Linux within Windows — VMware Player	57
5.2	DOS within Windows	57
5.3	Command line arguments	58
5.4	The UNREGISTERED banner	59
5.5	Title bar	59
5.6	Saving, Loading and Printing the DDLab screen	59
5.6.1	Saving and Loading the screen in DDLab’s own format	59
5.6.2	Saving and Printing the screen in Linux-like systems	60
5.6.3	Printing the screen in DOS	62
5.7	DDLab version and graphics info	62
5.8	The mouse pointer in DOS	62
5.9	Memory (DOS only)	63
5.9.1	Virtual Memory	63
5.10	DDLab prompts, and backtrack	63
5.10.1	Prompts: main sequence and pop-up windows	64
5.11	Exit DDLab	64
<b>6</b>	<b>The first prompt in DDLab</b>	<b>65</b>
6.1	TFO-mode, SEED-mode, FIELD-mode	65
6.2	The first prompt	66
6.2.1	TFO-mode: totalistic forwards-only	66
6.2.2	FIELD, or a run requiring a SEED	66
6.2.3	Notice of the current mode	66
6.2.4	The exLimits option	67
6.3	Graphics setup	67
6.3.1	The DDLab screen resolution, Linux-like systems	68
6.3.2	The DDLab screen resolution in DOS	68
6.3.3	White or black background	69

6.3.4	The color palette	69
6.4	Changing the font size and line spacing	69
6.4.1	Font size and line spacing, Linux-like systems	69
6.4.2	Font size, DOS	69
6.4.3	Line spacing, DOS	69
6.5	Changing the flashing cursor speed	70
6.6	Random number seed	70
<b>7</b>	<b>Value-range <math>v</math>, and <math>n, k</math> limits</b>	<b>71</b>
7.1	The value-range prompt	71
7.1.1	extending $k_{Lim}$ with exLimits	72
7.2	Limits on neighborhood size, $k_{Lim}$	72
7.2.1	TFO-mode with high $v, k$	72
7.3	Limits on network size for FIELD-mode, $n_{Lim}$	74
7.4	exLimits risks and insufficient memory	74
7.5	Limits on network size — exhaustive algorithm, $n_{exhL}$	74
<b>8</b>	<b>1d network size <math>n</math>, or range-<math>n</math></b>	<b>75</b>
8.1	Setting range of sizes, 1d	75
8.2	Setting the size of one network, 1d	76
8.3	Network size limits	78
8.4	Computational and speed limitations for attractor basins	78
8.4.1	Times for basin of attraction fields	79
8.4.2	Examples for 1d CA	81
8.4.3	Examples for RBN and DDN	82
<b>9</b>	<b>Neighborhood, <math>k</math>, or mixed-<math>k</math></b>	<b>83</b>
9.1	Selecting $k$ , or a $k$ -mix, for 1d networks	83
9.1.1	$k_{Lim}$ and minimum $k$	84
9.2	Effective $k=0$	84
9.3	Specifying the $k$ -mix	84
9.4	Loading a $k$ -mix file	85
9.5	Setting the $k$ -mix by hand	85
9.6	Setting a normal distribution	85
9.7	Specify each $k$ , or power-law distribution	86
9.7.1	Specify the percentage of different $k$	86
9.7.2	Setting a power-law distribution of $k$	86
9.7.3	Showing the distribution	87
9.8	Setting the $k$ -mix at random	87
9.9	Setting a $k$ -mix with uniform $k$	87
9.10	Increasing $k_{max}$	88
9.11	Reviewing the $k$ -mix	88
9.11.1	Reviewing the $k$ -mix in large networks	89
9.11.2	Jumping to a new cell index	89
9.11.3	Saving the $k$ -mix file	89
9.12	Reviewing the $k$ -mix within “network architecture”	89
<b>10</b>	<b>The local neighborhood, and network geometry</b>	<b>90</b>
10.1	The CA neighborhood	90
10.1.1	The pseudo-neighborhood, RBN and DDN	90
10.1.2	1d neighborhood	91
10.1.3	2d neighborhood	91

10.1.4	3d neighborhood	93
10.2	Network geometry	94
10.2.1	1d network indexing	95
10.2.2	2d network indexing	95
10.2.3	3d network indexing	96
<b>11</b>	<b>Setting the wiring, quick settings</b>	<b>97</b>
11.1	The first wiring prompt	97
11.2	Local 1d wiring	97
11.3	Special wiring	97
11.4	Loading the wiring scheme	98
11.5	Random 1d wiring	98
11.6	2d or 3d wiring	98
11.6.1	Setting 2d and 3d network size — SEED/TFO-mode	98
11.6.2	Setting 2d and 3d network size — FIELD-mode	99
11.7	Set $k$ , or $k$ -mix	100
11.8	Reviewing wiring, after quick settings	100
<b>12</b>	<b>Setting special wiring</b>	<b>101</b>
12.1	Setting up the network geometry	101
12.2	Hypercube wiring	102
12.2.1	degrading the hypercube	102
12.3	1d, 2d and 3d special wiring	103
12.3.1	Random 2d and 3d	103
12.4	Special wiring, local	104
12.4.1	Local 1d treated as random	104
12.4.2	Local 2d and 3d	104
12.5	Special wiring, random	104
12.5.1	Applying wiring biases to parts of the network	105
12.5.2	Confining random wiring to a set zone	105
12.5.3	Setting local wiring as random	105
12.5.4	Release wires from zone	106
12.5.5	Suppress periodic boundary conditions	106
12.5.6	Self-wiring	106
12.5.7	Distinct wiring	107
12.5.8	Suppress links to 3d layers	107
12.5.9	Force a direct link to a 3d layer	107
12.5.10	Force random links to specific 3d layers	107
12.5.11	Same random wiring everywhere	108
12.6	Wiring by hand	108
12.7	Reviewing wiring	109
<b>13</b>	<b>Rule types</b>	<b>110</b>
13.1	Selecting the rule type	110
13.1.1	Select full rule-tables, rcode	110
13.1.2	Select kcode or tcode in TFO-mode	111
13.2	Rule types and combinations	111
13.2.1	Rule-table size implications	112
13.3	Binary full rule-table — rcode	112
13.3.1	The binary neighborhood matrix	113
13.4	Binary totalistic rules	114

13.5	Multi-value full rule-table — rcode	114
13.5.1	The multi-value neighborhood matrix	115
13.6	Multi-value totalistic rules, tcode and kcode	115
13.6.1	k-totalistic rules - kcode	116
13.6.2	kcode $v=3$ matrix	117
13.6.3	t-totalistic rules - tcode	117
13.7	Outer-totalistic rules	118
13.8	Reaction-Diffusion dynamics	118
13.8.1	Reaction-Diffusion from outer-kcode	119
13.8.2	Reaction-Diffusion from a full rule-table — rcode	120
13.8.3	Selecting the threshold interval	120
<b>14</b>	<b>Rulemix options</b>	<b>121</b>
14.1	Single rule or rulemix, and other options	121
14.1.1	Summary of rulemix and other options	122
14.1.2	Density-bias ( $\lambda$ -parameter)	122
14.2	Outer-totalistic kcode or tcode	123
14.2.1	Setting reaction-diffusion by outer-kcode	123
14.2.2	The game-of-Life and other Life-like rules by outer-kcode	123
14.3	Setting the neighborhood, $k=0$	125
14.4	Methods for setting the rulemix or rule-subset	125
14.4.1	Setting a rulemix directly	126
14.4.2	Setting a rulemix indirectly - specify a rule-subset	126
14.4.2.1	Setting a rulemix indirectly - apply the rule-subset	127
14.4.3	A rulemix with just one rule	127
14.5	Rulemix - random	127
14.6	Rulemix by hand	128
14.6.1	By hand reminder	128
14.6.2	By hand single rule prompt	129
14.6.3	By hand options	129
14.6.4	Change the selection method or rule index	129
14.6.5	Complete the rulemix automatically	130
14.6.6	Copy rules automatically for a $k$ -mix	130
14.6.7	Mixed $k$ where all $k$ 's (and rules) are the same	130
14.7	Rulemix - majority	131
14.8	Rulemix - majority with shifted uniform outputs	131
14.9	Rulemix for large networks, or large $k$	132
14.10	The all 0s output	132
14.11	Amending the neighborhood matrix	132
14.12	List Post functions	133
14.12.1	Initial Post-function prompt	133
14.12.2	Restrict Post-functions	134
14.12.3	Set Post-function sample size	134
14.12.4	Final Post-function prompt	134
<b>15</b>	<b>Setting Canalyzation in a random rcode-mix</b>	<b>135</b>
15.1	Selecting Canalyzing	135
15.1.1	Selecting canalyzing from the rcode-mix	135
15.1.2	Selecting canalyzing from wiring graphic — transform rule	136
15.1.3	Selecting canalyzing from the Derrida plot	136
15.2	The first canalyzing prompt	136

15.3	Canalyzing percentage or number . . . . .	136
15.4	Canalyzing - homogeneous- $k$ . . . . .	137
15.5	Canalyzing - mixed- $k$ . . . . .	138
15.5.1	Canalyzing for the whole network - mixed- $k$ . . . . .	139
15.5.2	Canalyzing for a particular $k$ in a mixed- $k$ network . . . . .	140
15.6	Canalyzing for large networks, or large $k$ . . . . .	141
<b>16</b>	<b>Setting a single rule</b> . . . . .	<b>142</b>
16.1	The first single rule prompt . . . . .	142
16.1.1	Single rcode prompt examples . . . . .	142
16.1.2	TFO-mode single rule prompt examples . . . . .	143
16.2	Methods for setting a rule . . . . .	143
16.3	Setting the rule at random . . . . .	144
16.3.1	Random rule density-bias ( $\lambda$ parameter) . . . . .	145
16.3.2	Rule value-bias . . . . .	146
16.3.3	Random rule parameters . . . . .	147
16.4	Setting the rule as bits or values . . . . .	148
16.4.1	Rules: bits/values reminder window . . . . .	148
16.4.2	Rule: bits/values current settings inset . . . . .	150
16.4.3	Rules: setting bits/values with the keyboard and mouse . . . . .	150
16.4.4	Rules: save as 1d seed or PostScript . . . . .	151
16.4.5	Rules: PostScript prompt . . . . .	151
16.5	Setting the rule in hex . . . . .	153
16.6	Setting the rule in decimal . . . . .	153
16.7	Setting a majority rule . . . . .	154
16.8	Majority with shifted uniform outputs . . . . .	156
16.9	Setting Altenberg rules . . . . .	156
16.10	The game-of-Life and other Life-like rules — rcode . . . . .	157
16.10.1	Setting Life-like rules — rcode . . . . .	158
16.11	Setting a chain-rule . . . . .	159
16.12	Setting reaction-diffusion — rcode . . . . .	160
16.13	Repeating the last rule . . . . .	161
16.14	Loading a single rule . . . . .	161
16.15	Automatic saving of last rule . . . . .	161
16.16	Single rule file encoding . . . . .	161
16.17	Create a similar kcode with increased $k$ . . . . .	162
16.18	Show the rule in the terminal . . . . .	163
16.18.1	Immediate rule data . . . . .	163
16.18.2	Immediate rule data for a rulemix by hand . . . . .	163
16.18.3	Rule data in more detail — vertical layout . . . . .	164
16.18.4	Additional rule data options for kcode . . . . .	164
16.18.5	Swapping kcode values . . . . .	165
16.19	The rule window . . . . .	165
16.19.1	Decoding the rule window . . . . .	166
16.20	Complementary values . . . . .	167
16.21	Transforming the single rule . . . . .	167
16.22	Saving/Loading a rule as an ASCII string . . . . .	168
16.22.1	ASCII rule encoding . . . . .	168
<b>17</b>	<b>Reviewing network architecture</b> . . . . .	<b>169</b>
17.1	The network architecture prompt . . . . .	169



17.2	The wiring matrix	171
17.2.1	Viewing the wiring matrix and creating vector PostScript	172
17.2.2	Amending the matrix	172
17.3	The wiring graphic	173
17.4	The wiring graphic reminder	174
17.4.1	Wiring graphic options summary	175
17.5	Wiring graphic, mouse pointer/click	177
17.6	Wiring graphic, 1d	178
17.6.1	Data — 1d wiring graphic	179
17.6.2	Moving or jumping between cells, 1d	180
17.6.3	Defining a block, 1d	180
17.6.4	Toggling the block, 1d	182
17.6.5	Include the pseudo-neighborhood, or direct wiring only, 1d	182
17.6.6	Recursive inputs to a cell, 1d	182
17.6.7	Recursive outputs from a cell, 1d	182
17.6.8	Untangling the wiring	185
17.6.9	Deleting a cell	185
17.7	Wiring graphic, 2d	186
17.7.1	Data — 2d wiring graphic	186
17.7.2	Moving or jumping between cells, 2d	187
17.7.3	Alternative wiring presentation, 2d	187
17.7.4	Include the pseudo-neighborhood, or direct wiring only, 2d	187
17.7.5	Defining a block, 2d	188
17.7.6	Toggling the block, 2d	189
17.7.7	Expand/Contract the scale, 2d	189
17.7.8	Shifting the 2d graphic up and down	189
17.8	Wiring graphic, 3d	190
17.8.1	Data — 3d wiring graphic	190
17.8.2	Moving or jumping between cells, 3d	191
17.8.3	Alternative wiring presentation, 3d	192
17.8.4	Include the pseudo-neighborhood, or direct wiring only, 3d	192
17.8.5	Defining a block, 3d	192
17.8.6	Toggling the block, 3d	193
17.8.7	Toggle 3d background grid	193
17.8.8	Expand/Contract the scale, 3d	194
17.8.9	Shifting the 3d graphic up and down	194
17.9	Further options for the 1d, 2d and 3d wiring graphics	195
17.9.1	Decoding wiring graphic data — 1d, 2d and 3d	195
17.9.2	Computing the (weighted) average $\lambda$ and $Z$ parameters	196
17.9.3	Options for learning pre-images	196
17.9.4	Hand rewiring	196
17.9.5	Random rewiring	197
17.9.6	Biased random rewiring	197
17.9.7	Local 1d, 2d or 3d wiring	198
17.9.8	Changing the neighborhood size, $k$	198
17.9.9	Kill a cell	198
17.9.10	Revising and copying the rule	199
17.9.11	Transforming the rule	199
17.9.12	Filing, from the wiring graphic	199
17.9.13	The histogram of the network's $k$ and output distribution	200
17.9.14	Creating a vector PostScript file of the wiring graphic	201

<b>18 Transforming rules</b>	<b>202</b>
18.1 Options for transforming rules	203
18.1.1 Transform options, single rule	203
18.1.2 Transform options, mixed- $k$	203
18.1.3 Transform options, mixed rule, homogeneous- $k$	203
18.1.4 Transform all cells in a mixed rule network	204
18.2 Saving or printing the transformed rule	204
18.3 Inverting the rcode	204
18.4 Solidifying the rule	205
18.5 Equivalence classes and rule clusters	205
18.5.1 Complementary transformation	205
18.5.2 Equivalent rcode by the Negative transformation	207
18.5.3 Equivalent rcode by the Reflection transformation	207
18.6 Setting canalizing inputs, single rcode	207
18.6.1 Canalizing inputs at random, single rcode	207
18.6.2 Canalizing inputs explicitly, single rcode	207
18.7 Neutral transformations of the neighborhood $k$	208
18.7.1 Equivalent rules with greater $k$	208
18.7.2 Reducing $k_{max}$ to the maximum $k$ in the network	208
18.7.3 Effective $k$	209
18.7.4 Reverse engineering - loading an exhaustive map	209
<b>19 File/print network architecture</b>	<b>210</b>
19.1 Network filing options	210
19.2 Wiring/rulemix filenames	211
19.3 Wiring/rulemix encoding	211
19.3.1 Mixed- $k$ encoding	212
19.4 Loading networks and sub-networks	213
19.4.1 loading networks — compatibility	213
19.4.1.1 Compatibility with $v$	213
19.4.1.2 Compatibility with $k$	214
19.4.1.3 Compatibility with $n$	214
19.4.1.4 Compatibility with edge sizes and dimensions	214
19.4.2 Loading a complicated network into a CA and vice versa	215
19.4.3 Loading sub-networks in a set position	216
19.4.4 Loading $k$ -mix networks	217
19.5 Saving just the $k$ -mix	218
19.6 Printing network data to the terminal or file	218
<b>20 The network-graph, and attractor jump-graph</b>	<b>221</b>
20.1 Unravelling the jump-graph and the network-graph	221
20.2 The network-graph	223
20.2.1 The network-graph reminder	223
20.3 The jump-graph of the basin of attraction field	224
20.3.1 Selecting the jump-graph	224
20.3.2 The jump-graph reminder	225
20.4 Initial graph options	227
20.5 Dragging nodes or fragments	229
20.5.1 The drag reminder	230
20.5.2 Drag graph options	232
20.5.3 Defining a block	234

20.6	Probabilistic “ant”	234
20.6.1	Show ant hits	235
20.7	Redraw basins at jump-graph nodes	236
20.7.1	PostScript of jump-graph basins	238
20.8	PostScript of the network-graph or jump-graph	239
20.9	Graph layout file	239
20.10	Revise settings	239
20.11	Unreachable nodes	240
20.12	The adjacency-matrix and jump-table	241
20.12.1	Printing and scanning tables	242
20.13	Space-time patterns within the network-graph	242
<b>21</b>	<b>The Seed or initial state</b>	<b>244</b>
21.1	The seed prompt	244
21.2	Methods for setting a seed	246
21.3	Setting the seed at random	247
21.3.1	Seed at random — further options	248
21.3.2	Non-zero seed density-bias	249
21.3.3	Seed or block value-bias	250
21.4	Setting the seed as bits or values	251
21.4.1	2d patch options	252
21.4.2	Seed: bits/values reminder window	252
21.4.3	Seed: bits/values options summary	253
21.4.4	Seed: bits/values current settings inset	256
21.4.5	Seed: setting bits/values with the keyboard and mouse	256
21.4.6	setting bits/values: 1d segments	257
21.4.7	setting bits/values: 1d shown as 2d	257
21.4.8	Setting bits/values: filing and PostScript	257
21.4.9	Setting bits/values: saving a patch	259
21.4.10	Setting bits/values: PostScript prompt	260
21.4.11	Setting bits/values: PostScript 3d image	261
21.5	Setting the seed in hex	262
21.6	Setting the seed in decimal	263
21.7	Loading a seed, and loading constraints	264
21.7.1	loading a seed — all parameters equal	264
21.7.2	loading a seed — unequal value-ranges	264
21.7.3	loading a seed that fits within the base	264
21.7.4	loading a seed that does not fit within the base, 2d and 3d	265
21.7.5	loading a seed that does not fit within the base, 1d	266
21.8	Saving a seed	267
21.9	Seed file encoding	267
21.10	Saving/Loading a seed as an ASCII string	268
21.10.1	ASCII seed encoding	269
21.11	Saving/Loading a seed in the Golly file format	269
21.11.1	The Golly file prompt	270
<b>22</b>	<b>The Derrida plot</b>	<b>271</b>
22.1	Selecting the Derrida plot	272
22.2	Derrida plot options	272
22.2.1	Derrida plot parameters	273
22.3	Data within the Derrida plot	274

22.3.1	Network data	274
22.3.2	Derrida data	274
22.4	Interrupting the Derrida plot	275
22.5	Completing the Derrida plot	275
22.5.1	Completing the Derrida plot parameters	275
22.5.2	Data at the end of each plot	276
22.6	The Derrida coefficient	276
22.7	Automatic plots of CA rule clusters	277
<b>23</b>	<b>Graphic conventions for attractor basins</b>	<b>280</b>
23.1	Basins of Attraction — the idea	280
23.2	Network states, nodes	281
23.3	Attractor cycles	283
23.4	Transient trees	283
23.4.1	Transient tree colors	283
23.4.2	Transient trees for uniform states	284
23.4.3	Subtree only	285
<b>24</b>	<b>Output parameters for attractor basins</b>	<b>286</b>
24.1	The first output parameter prompt for attractor basins	286
24.1.1	Output parameter prompt for attractor basins — options summary	287
24.2	PostScript of attractor basins	288
24.3	Activate the jump-graph	288
24.4	Miscellaneous (hard to categorize) options	289
24.5	State-space matrix	289
24.5.1	State-space matrix — options summery	290
24.5.2	Toggle the matrix on-the-fly	292
24.6	In-degree frequency histogram	292
24.6.1	In-degree frequency cut-off	292
24.6.2	Drawing the in-degree histogram	293
24.6.3	In-degree data and prompts	295
24.6.4	in-degree window — data decode	295
24.6.5	in-degree window — options summery	295
24.6.6	In-degree log plot	296
24.6.7	Rescaling the x/y-axis	296
24.7	Density classification problem — attractor basins	297
24.8	Screen-saver demo	298
24.9	$G$ -density, $Z$ and $\lambda$	298
24.9.1	$G$ -density, $Z$ and $\lambda$ — options summery	300
24.9.2	$G$ -density plotted against network size	300
24.9.3	$G$ -density against $Z$ and/or $\lambda_{ratio}$	301
24.9.3.1	$G$ -density for tcode, or rcode $k \leq 3$	302
24.9.3.2	$G$ -density for $k > 3$ rcode	302
24.9.4	$Z - \lambda_{ratio}$ plots	302
24.9.4.1	$Z - \lambda_{ratio}$ or $Z_{left} - Z_{right}$	303
24.9.4.2	$Z - \lambda_{ratio}$ or $Z_{left} - Z_{right}$ for $k \geq 4$ rcode	303
24.9.4.3	$Z - \lambda_{ratio}$ or $Z_{left} - Z_{right}$ for tcode	304
24.9.5	Proportions of canalizing inputs and $\lambda_{ratio}-P$	305
24.10	“Backwards” space-time patterns	306
24.10.1	Scroll space-time patterns	306
24.11	Basin speed	308

24.12	Basin on-the-fly options	308
<b>25</b>	<b>Layout of attractor basins</b>	<b>309</b>
25.1	The layout preview	309
25.2	The first layout prompt	310
25.2.1	Reset all layout defaults	311
25.2.2	Reset transient scale	311
25.2.3	Mutants for single basins on one screen	313
25.2.4	Basin scale, attractor radius	313
25.2.5	Basin start position	313
25.2.6	Show the field as successive basins	313
25.2.7	Basin spacing and stagger rows	313
25.2.8	Select minimum right border width	314
25.2.9	Amend the spacing increase for a range of $n$	314
25.2.10	Accept or revise layout parameters	315
25.3	Amend the layout during pause	315
25.3.1	Amend the orientation and fan angle during pause	315
25.3.2	Amend the spacing and right border during pause	315
25.3.3	Amend the next position during pause	316
25.3.4	Amend the spacing increase during pause	317
<b>26</b>	<b>Display of attractor basins</b>	<b>318</b>
26.1	Attractor basins with null boundary conditions	318
26.2	Compression of equivalent CA dynamics for periodic boundaries	319
26.2.1	Deactivate compression	321
26.2.2	Pre-images of uniform states	321
26.2.3	Suppress copies of trees (and subtrees)	323
26.3	Node display	323
26.3.1	Node colors	325
26.3.2	Highlight attractor, or subtree root, in 2d	326
26.3.3	Change the 2d node $i, j$	327
26.3.4	Alter scale, divisions and dots — node as bits/values	327
26.3.5	Alter decimal or hex node scale	327
26.4	Change orientation, fan angle, edge color	328
26.4.1	Orientation	328
26.4.2	Pre-image fan angle	328
26.4.3	Edge color	329
26.5	Limiting backward steps	330
<b>27</b>	<b>Pausing attractor basins, and data</b>	<b>331</b>
27.1	Pause stages hierarchy	331
27.1.1	Pause after each field for a range of fields	332
27.1.2	Pause after each basin, tree, or pre-image fan	332
27.1.3	The pause prompt	332
27.2	Attractor basin complete — data window	332
27.2.1	Data on basin of attraction fields	333
27.2.2	Errors in basin of attraction fields	334
27.2.3	Data on basins	334
27.2.4	Data on trees	335
27.2.5	Data on pre-image fans	335
27.2.6	Data on subtrees	336

27.2.7	Data on subtrees from a uniform state	337
27.3	Print or save data	339
27.4	Data format	340
27.4.1	Network parameters data	340
27.4.2	Basin field data	340
27.4.3	Key to basin data order	341
27.4.4	Tree data	341
27.4.5	Key to tree data order	342
27.4.6	Single basin data	343
27.4.7	Subtree data	343
27.5	List of states	344
27.5.1	Just the list of states	344
27.5.2	The list of states with basin data	345
27.5.3	The list of states with basin <i>and</i> tree data	346
<b>28</b>	<b>Mutation of attractor basins</b>	<b>347</b>
28.1	The first mutation prompt	347
28.1.1	The first mutation prompt — options summary	347
28.2	Mutate wiring	348
28.3	Special mutation options	350
28.3.1	Bit-flip or value-flip in sequence	352
28.4	Flip bits or values	353
28.5	No pause before next mutant	354
<b>29</b>	<b>Final options for attractor basins</b>	<b>355</b>
29.1	Subtree or single basin	355
29.2	Subtree: Run forward before running backwards	356
29.3	Single basin of attraction	356
29.3.1	Single basin history limit	356
29.3.2	Interrupting while looking for attractor	356
29.4	Final attractor basin prompt	357
29.4.1	Final attractor basin prompt — conditions and reminders	357
29.4.2	Final attractor basin prompt — options summary	357
29.5	Limit backward steps	358
29.5.1	The bare attractor — limit backward steps to zero	359
29.6	Viewing the partial pre-image stack	361
29.6.1	Partial pre-image stack, local wiring	362
29.6.2	Partial pre-image stack, nonlocal wiring	362
29.6.3	Reorder to optimize the nonlocal reverse algorithm	362
29.7	Exhaustive algorithm	364
29.7.1	Generating the exhaustive list	365
29.7.2	Saving the exhaustive pairs	366
29.7.3	Printing the exhaustive pairs in the terminal	366
29.8	Random map	367
29.8.1	Loading the random map or exhaustive pairs	369
29.8.2	Biasing the random map by Hamming distance	369
29.8.3	Random map data	369
29.9	Sequential updating	370
29.9.1	Random order	370
29.9.2	Set specific order	371
29.9.3	List all orders	372

29.9.4	List all orders — set order seed . . . . .	373
29.9.5	Drawing the attractor basin with sequential updating . . . . .	373
29.10	Neutral order components . . . . .	373
29.10.1	Neutral subtree . . . . .	375
29.10.2	Neutral field . . . . .	376
<b>30</b>	<b>Drawing attractor basins, and changes on-the-fly</b>	<b>378</b>
30.1	The progress bar for basin of attraction fields . . . . .	378
30.2	Attractor basins, interrupting and changing . . . . .	379
30.2.1	Abandoning a tree and continuing with the next tree . . . . .	380
30.2.2	Errors in attractor basins . . . . .	381
30.2.3	Abandon the attractor basin . . . . .	381
30.3	Attractor basin options, on-the-fly . . . . .	381
30.4	Attractor basin complete prompt . . . . .	382
30.5	Further attractor basin complete options . . . . .	383
30.5.1	Attractor basin — revising rule/s . . . . .	384
30.5.2	Attractor basin - revising the seed . . . . .	384
<b>31</b>	<b>Output parameters for space-time patterns</b>	<b>386</b>
31.1	The first output parameter prompt for space-time patterns . . . . .	386
31.1.1	Output parameter prompt for space-time patterns — options summary . . . . .	387
31.2	Miscellaneous options — space-time patterns . . . . .	388
31.2.1	Color cells by value or neighborhood . . . . .	388
31.2.2	State-space matrix and return map . . . . .	389
31.2.2.1	State-space matrix . . . . .	389
31.2.2.2	Return map by value . . . . .	390
31.2.3	Frozen generation size . . . . .	391
31.2.4	Cell scale . . . . .	391
31.2.5	Space-time patters in other than the native dimension . . . . .	391
31.2.6	Scrolling 1d space-time patterns . . . . .	391
31.2.7	Pause and step . . . . .	392
31.2.8	Speed of iteration . . . . .	392
31.2.9	Glider rule order . . . . .	392
31.2.10	Inverting the kcode in TFO-mode . . . . .	393
31.3	Periodic or Null boundary conditions . . . . .	393
31.4	Asynchronous and noisy updating . . . . .	393
31.4.1	Probabilistic updating . . . . .	394
31.4.2	sequential updating — space-time patterns . . . . .	395
31.4.3	partial order updating — space-time patterns . . . . .	396
31.5	Input-entropy and pattern density . . . . .	397
31.5.1	Single cell input-entropy and pattern density . . . . .	397
31.5.2	Generation size - moving window of time-steps . . . . .	398
31.5.3	On-the-fly changes to input-entropy and pattern density . . . . .	399
31.6	Damage, the difference between two networks . . . . .	399
31.6.1	Duplicate the network and seed . . . . .	401
31.6.2	The Damage Histogram . . . . .	402
31.6.3	Drawing the damage histogram . . . . .	403
31.6.4	Pausing the damage histogram . . . . .	404
31.7	Attractor histogram . . . . .	406
31.7.1	Density classification problem — attractor histogram . . . . .	407
31.7.2	Drawing the attractor histogram . . . . .	407

	31.7.2.1	Histogram window information . . . . .	407
31.7.3		Pausing the attractor histogram . . . . .	408
31.7.4		Rescaling the attractor histogram . . . . .	411
31.7.5		Attractor histogram data . . . . .	411
	31.7.5.1	Attractor histogram screen data . . . . .	411
	31.7.5.2	Attractor histogram data decode . . . . .	412
	31.7.5.3	Histogram data for density rules . . . . .	412
31.7.6		Print/Save attractor state data . . . . .	412
31.7.7		Sorting the attractor histogram . . . . .	413
31.7.8		Attractor histogram jump-graph . . . . .	415
31.8		Skeleton (fuzzy attractor) histogram . . . . .	416
	31.8.1	Skeleton parameters prompt . . . . .	416
	31.8.2	Drawing the Skeleton histogram . . . . .	418
	31.8.3	Pausing the skeleton histogram . . . . .	419
	31.8.4	Skeleton histogram data . . . . .	420
		31.8.4.1 Skeleton histogram data decode . . . . .	421
	31.8.5	Sorting the skeleton histogram . . . . .	421
<b>32</b>		<b>Drawing space-time patterns, and changes on-the-fly</b>	<b>422</b>
32.1		On-the-fly key index . . . . .	422
32.2		On-the-fly prompts — bottom title bar . . . . .	424
32.3		Summary of on-the-fly options for space-time patterns . . . . .	424
32.4		Updating (tog) . . . . .	430
	32.4.1	U/Y..sync-seq/sync-porder . . . . .	430
	32.4.2	{/}..prob:update1-0.95/output1-0.95 . . . . .	430
32.5		Change rule . . . . .	430
	32.5.1	r,R/K/k..rnd/vrnd/tcode/kcode . . . . .	431
	32.5.2	[/].iso/togflip(on) . . . . .	432
	32.5.3	O/A/M/C..Orig/Alt/Maj/Chain . . . . .	432
	32.5.4	1/2..rnd bitflip/restore . . . . .	432
	32.5.5	Z/z..force Z higher/lower . . . . .	433
	32.5.6	b/B..flip all0s->0/allVs->V . . . . .	433
	32.5.7	8..rulemix-single . . . . .	433
32.6		Rule samples . . . . .	434
	32.6.1	g..load glider rule (rnd) . . . . .	434
		32.6.1.1 creating a glider rule collection . . . . .	434
	32.6.2	V..load, jmp, scan . . . . .	435
	32.6.3	w/:/9..next/prev/rnd . . . . .	437
	32.6.4	uE..create sample . . . . .	437
32.7		Change wiring . . . . .	438
	32.7.1	m/W..move 1 wires . . . . .	438
		32.7.1.1 7..nonlocal-local . . . . .	438
	32.7.2	..periodic-null . . . . .	438
32.8		Change seed/size . . . . .	439
	32.8.1	4/v..rnd seed/block . . . . .	439
	32.8.2	l/L..rnd value/block . . . . .	439
	32.8.3	o/~..original/last . . . . .	440
	32.8.4	5/6..singleton pos/neg . . . . .	440
	32.8.5	5/6..singleton zero/rnd . . . . .	440
	32.8.6	N/n..inc/decrease 1 cell . . . . .	440
32.9		Presentation . . . . .	440



32.9.1	x..tog slant(off)	440
32.9.2	x..tog hex(off)	441
32.9.3	i!/..tog divs(off)	441
32.9.4	@..tog balls outline	442
32.9.5	3/./^..incolor/dot/bground	442
32.9.6	d/-.colors:swap/black-blue	443
32.9.7	d/-.tog shuffle colors/restore	443
32.9.8	S..tog space-time display	443
32.9.9	P..tog skip steps=1 (off)	443
32.9.10	\$..tog sound	443
32.9.11	e/c..expand/contr scale	443
32.10	1d 2d 3d	444
32.10.1	T..tog 1d-2d-3d	445
32.10.2	t..tog 2d-2d+time	446
32.10.3	L..tog balls	446
32.10.4	p..plane	447
32.10.5	J..invisible	447
32.11	Frozen/Filter	447
32.11.1	h..nor-dy-f1-f2-bin	447
32.11.2	H..f-gens (now 20)/bins(10)	450
32.11.3	frozen generations	450
32.11.4	frequency bins	450
32.11.5	f/F/a..filter/undo/all	451
32.12	Analysis	453
32.12.1	0/%..tog lookuphist:1-2/1-time	455
32.12.2	)/(..lookhist: amplify/restore	455
32.12.3	s..tog entropy-density	456
32.12.4	j..tog ent-in-both	456
32.12.5	u..tog entropy/density plot	457
32.12.6	G..a-gens (now 10)	457
32.12.7	D..return map by value	457
32.12.8	;.return map by density	457
32.12.8.1	return map by density — saving the data	458
32.12.9	y..state-space	459
32.12.10	=..tog diff (keep damage)	459
32.13	Miscellaneous	459
32.13.1	X..index display	459
32.13.2	*..tog end pause (on)	459
32.13.3	#..tog scrolling	459
32.13.3.1	1d scrolling	460
32.13.3.2	2d diagonal scrolling	460
32.13.3.3	network-graph scrolling	460
32.13.3.4	2d+time scrolling	460
32.13.4	+..tog time-step pause	460
32.13.5	</>..slow/max speed	461
32.13.6	q..pause	461
32.14	Interrupting space-time patterns	461
32.15	Rule details for space-time patterns	461
32.16	Space-time pattern interrupt/pause prompt	462
32.16.1	Revising rule/net	463
32.16.2	Classified samples of rule-space — load/keep	464

32.16.3	Directly scanning for PostScript . . . . .	464
32.16.4	Revising the seed and native PostScript . . . . .	464
32.16.5	Space-time patterns on the network-graph . . . . .	466
32.16.6	Fixed borders . . . . .	466
32.16.7	Finer control of filtering . . . . .	466
32.16.8	Space-time — skip, pause, step and speed . . . . .	467
32.16.9	Miscellaneous options . . . . .	468
32.17	Quit and further options . . . . .	468
32.18	Directly scanning STP for vector PostScript . . . . .	469
32.19	Network-graph layout of space-time patterns . . . . .	471
32.19.1	On-the-fly options within the network-graph . . . . .	473
32.19.2	Network-graph space-time pattern as vector PostScript . . . . .	474
<b>33</b>	<b>Classifying rule space</b> . . . . .	<b>475</b>
33.1	Input entropy and variability (min-max or sdev) . . . . .	478
33.2	Creating a rule sample — initial prompts . . . . .	479
33.2.1	Biasing random rules . . . . .	480
33.3	Running a test . . . . .	480
33.3.1	Test data . . . . .	481
33.3.2	Test options . . . . .	482
33.4	Creating an automatic rule sample . . . . .	482
33.5	Loading, sorting and displaying a sample . . . . .	484
33.6	The rule sample scatter plot . . . . .	486
33.6.1	Probing the scatter plot with the mouse, and selecting rules . . . . .	486
33.6.2	Defining a scatter plot patch with the mouse or keyboard . . . . .	486
33.6.3	The scatter plot as a 2d frequency histogram . . . . .	488
33.7	Scanning sample space-time patterns . . . . .	489
33.7.1	Scanning on-the-fly . . . . .	490
33.7.2	Scanning automatically in blocks of time-steps . . . . .	491
33.7.3	Scanning 1d samples . . . . .	491
33.7.4	Scanning 2d or 3d samples . . . . .	493
33.8	Listing a sample . . . . .	494
33.8.1	Selecting a rule from the list . . . . .	495
33.9	Listing by plot coordinates or probing with the mouse . . . . .	495
33.10	Rule sample encoding . . . . .	496
<b>34</b>	<b>Learning, forgetting, and highlighting</b> . . . . .	<b>497</b>
34.1	Learning/forgetting methods . . . . .	497
34.1.1	Highlighting states in attractor basins . . . . .	498
34.1.2	Basin layout for learning . . . . .	498
34.2	Selecting the learn/forget/highlight window . . . . .	500
34.2.1	Selecting the wiring graphic . . . . .	501
34.2.2	Selecting the network architecture prompt . . . . .	501
34.3	Select the target state . . . . .	501
34.4	Select aspiring pre-image/s . . . . .	502
34.4.1	Odd or even parity . . . . .	503
34.4.2	Range of decimal equivalents . . . . .	503
34.4.3	Pre-images according to Hamming distance . . . . .	503
34.4.4	List of aspiring pre-images . . . . .	504
34.4.5	Review target state and aspiring pre-images . . . . .	504
34.5	Learn, forget, or highlight only . . . . .	504

34.5.1	1d CA — local wiring and highlighting . . . . .	505
34.5.2	Nonlocal wiring — learn, forget, or highlight only . . . . .	505
34.5.3	Learning/forgetting by wire moves or bit/value-flips . . . . .	506
34.6	Learning/forgetting by bit/value-flips . . . . .	506
34.7	Learning/forgetting by wire-moves . . . . .	507
34.8	Highlighting options . . . . .	510
34.9	Learning/forgetting/highlighting complete . . . . .	510
<b>35</b>	<b>Filing</b> . . . . .	<b>511</b>
35.1	File naming constraints in DDLab . . . . .	511
35.2	File types, default filenames, and extensions . . . . .	511
35.3	The filing prompt . . . . .	514
35.4	Loading constraints and warnings . . . . .	515
35.5	Changing the directory . . . . .	515
35.6	List files . . . . .	516
<b>36</b>	<b>Vector PostScript capture of DDLab output</b> . . . . .	<b>517</b>
36.1	Cropping and editing vector PostScript . . . . .	518
36.1.1	Cropping the PostScript image . . . . .	518
36.1.2	Amending the width of lines . . . . .	518
<b>37</b>	<b>Glossary</b> . . . . .	<b>519</b>
	<b>References</b> . . . . .	<b>525</b>
	<b>Index</b> . . . . .	<b>529</b>

# List of Figures

1	The basin of attraction field of a 3-value Cellular Automaton . . . . .	vii
2	A single basin of attraction . . . . .	xii
1.1	The basin of attraction field of a Cellular Automaton . . . . .	2
1.2	Themes in DDLab . . . . .	3
1.3	The space-time pattern of a 1d complex CA with interacting gliders . . . . .	6
1.4	The basin of attraction field of a complex CA . . . . .	7
1.5	A detail of a basin of attraction . . . . .	7
2.1	Cell value color scheme . . . . .	12
2.2	2d lattice with a hex/triangular lattice . . . . .	13
2.3	Drawing a 2d seed . . . . .	15
2.4	Space-time patterns of a 1d CA . . . . .	17
2.5	The basin of attraction field of a random Boolean network . . . . .	18
2.6	A basin of attraction of a random Boolean network . . . . .	19
2.7	Order–chaos measures for RBN . . . . .	22

2.8	1d CA space-time patterns showing ordered, complex and chaotic dynamics . . . . .	23
2.9	Subtrees of ordered-complex-chaotic CA . . . . .	25
3.1	2-way glider-gun, 2d space-time . . . . .	31
3.2	Byl's self reproducing loop . . . . .	32
4.1	A basin of attraction field . . . . .	36
4.2	A basin of attraction field, multi-value . . . . .	36
4.3	Backwards space-time patterns . . . . .	37
4.4	The state-space matrix . . . . .	37
4.5	Basin of attraction fields for a range of network size . . . . .	38
4.6	Examples of single basins . . . . .	39
4.7	A subtree . . . . .	40
4.8	A 1d space-time pattern . . . . .	42
4.9	A 1d CA shown as a scrolling ring of cells. . . . .	44
4.10	A 1d CA and filtered Altenberg rule . . . . .	45
4.11	Space-time patterns of the 2d game-of-Life . . . . .	46
4.12	Space-time snapshots of the 2d game-of-Life — frozen options . . . . .	47
4.13	Space-time patterns of the 2d game-of-Life scrolling diagonally . . . . .	48
4.14	2d space-time patterns, $v2k7$ CA on a hexagonal grid . . . . .	49
4.15	2d space-time patterns, $v3k6$ $k$ -totalistic 2d CA . . . . .	50
4.16	Examples of 3d $v2k7$ CA . . . . .	51
4.17	The screen-saver . . . . .	52
5.1	Typical graphical user interface when starting DDLab . . . . .	57
5.2	The main sequence of prompts . . . . .	61
6.1	A very small DDLab screen in Linux . . . . .	68
7.1	Value color key, $v = 2$ to $8$ . . . . .	71
8.1	Basin of attraction fields for a range of network size . . . . .	76
8.2	Single basins for a range of network size . . . . .	77
8.3	Subtrees for a range of network size . . . . .	77
8.4	A subtree for a 1d CA chain-rule . . . . .	79
8.5	Basin of attraction fields for a range of network sizes, $k=5$ . . . . .	80
8.6	CA Basin of attraction field, $v2k3$ rcode (dec)110, $n=31$ . . . . .	80
8.7	Basin of attraction fields for a range of network sizes, RBN, $v2k3$ . . . . .	81
9.1	A normal $k$ distribution . . . . .	85
9.2	A power-law $k$ distribution . . . . .	87
9.3	An example $k$ -mix . . . . .	88
9.4	$k$ -mix, 1d wiring graphic, time-steps . . . . .	89
10.1	1d neighborhood templates . . . . .	91
10.2	2d neighborhood templates . . . . .	92
10.3	3d neighborhood templates and indexing . . . . .	93
10.4	3d neighborhoods displayed in 2d . . . . .	94
10.5	1d network indexing . . . . .	95
10.6	2d network indexing . . . . .	95
10.7	3d network indexing . . . . .	96
10.8	3d network indexing, $i, j, h=1$ . . . . .	96

12.1	Hypercube and degraded hypercube	102
12.2	Hypercubes for $n \geq 32$	103
12.3	Confining 1d random wiring within a local zone	105
12.4	Confining 2d $k=9$ random wiring within a local zone	106
12.5	Confining 3d random wiring within a local zone	106
12.6	Setting wiring by hand on the blank wiring matrix	108
13.1	The binary neighborhood neighborhood matrix $k=1$ to 9	113
13.2	Multi-value neighborhood matrix examples	115
13.3	Reaction-diffusion dynamics, 2d	119
13.4	Reaction-diffusion dynamics, 3d	119
14.1	Glider guns in $v=8$ Life	124
14.2	Effective neighborhood $k=0$	125
14.3	2d CA perturbed by a chaotic block of rules	127
14.4	Shifted majority kcode-mix with random wiring	131
14.5	Multi-value neighborhood matrix, part only	132
15.1	Canalyzing frequency/saturation - homogeneous- $k$	137
15.2	Canalyzing saturation - 2d, mixed- $k$	138
15.3	Canalyzing - 2d, mixed- $k$ , for just one $k$	139
15.4	Canalyzing for large networks, large $k$	140
16.1	$v8k4$ kcode according to the default rule value-bias	146
16.2	Setting bits starting with all 0s	148
16.3	Setting values — alternative presentations	149
16.4	Drawing bits or values on the rcode pattern	151
16.5	Setting rcode in hex	153
16.6	Majority rcode $v2k9$	154
16.7	Majority kcode $v8k6$	155
16.8	Majority tcode $v8k7$	155
16.9	Flipped $v=2$ majority rcode with random wiring	156
16.10	Altenberg kcode $v8k7$	156
16.11	The game-of-Life rule	157
16.12	Fredkin's replicator	157
16.13	Glider guns in $v=3$ Life	158
16.14	Encryption with chain-rules	159
16.15	Encryption with chain-rules, multi-value	160
16.16	Analogous kcode with increased $k$	162
16.17	Rule window examples	166
16.18	Equivalent kcode - two values swapped, $v3k5$	167
17.1	Wiring matrix examples	171
17.2	The 1d wiring graphic, time-steps	173
17.3	The 1d circle wiring graphic	178
17.4	The 1d graphic, showing wiring to a block	179
17.5	The 1d graphic, showing the wiring of the whole network	180
17.6	Include the pseudo-neighborhood or direct wiring, 1d	181
17.7	Recursive inputs (direct and indirect) to a cell	183
17.8	Recursive outputs (direct and indirect) from a cell	184
17.9	Untangling the wiring	185
17.10	The 2d wiring graphic, pseudo-neighborhood, square, hex, direct	186

17.11	Alternative ways of showing cell connections, 2d	187
17.12	2d block, solid alternative	188
17.13	2d block, alternative presentations	189
17.14	2d wiring graphic of a large network	190
17.15	3d wiring graphic	191
17.16	Alternative ways of showing cell connections, 3d	192
17.17	3d block, solid alternative	193
17.18	3d block, alternative presentations	194
17.19	3d wiring graphic of a large network	195
17.20	Hand rewiring a single cell	197
17.21	Unbiased random wiring, 2d and 3d	197
17.22	Histograms of a power-law distribution of network links	200
18.1	Transforming rcode	204
18.2	Equivalence class — rule 193	206
19.1	Loading 3d network into 1d	213
19.2	Loading a 3d CA into a 2d CA	215
19.3	Loading a DDN into a CA	215
19.4	Loading a 2d CA into a larger 2d CA	216
19.5	Loading a small 3d DDN into a 3d DDN	217
19.6	Space-time snapshot of a 2d CA inside another 2d CA with different $k$	218
20.1	Simple network-graphs	222
20.2	Power-law network-graphs)	223
20.3	The jump-graph + basins	225
20.4	Screen shot of the basin of attraction field and jump-graph	226
20.5	Dragging the jump-graph, further examples	227
20.6	Unscrambling the network-graph of a scale-free RBN	228
20.7	Dragging network-graph nodes and fragments — 1d	230
20.8	Dragging network-graph nodes and fragments — 2d	230
20.9	Dragging network-graph nodes and fragments — 3d	231
20.10	Probabilistic ant hits	236
20.11	Inserting basins in the jump-graph	237
20.12	Basins of attraction at the jump-graph nodes	238
20.13	Disconnecting and isolating unreachable nodes	240
20.14	A 2d CA shown as a scrolling ring of cells.	243
21.1	Seed array indexing, 1d	244
21.2	Seed array indexing, 2d	245
21.3	Seed array indexing, 3d	245
21.4	Setting a seed at random	248
21.5	Setting the density-bias	250
21.6	Playing with a 2d active patch	252
21.7	Divisions between cells	254
21.8	1d seed aspect ratio	254
21.9	Drawing bits or values on a 2d seed	256
21.10	Examples of PostScript seed output	258
21.11	Symbols for PostScript seed output	260
21.12	3d bit/value seed	262
21.13	Loading a 2d seed into a 3d network	262
21.14	Setting the seed in hex	263

21.15	Loading 3d and 2d seeds that fit into a 3d base network	265
21.16	Loading seed, 3d to 2d, and 2d to smaller 2d	266
21.17	Loading seed, 1d to 2d and 3d	267
22.1	Derrida plots	272
22.2	The Derrida coefficient	277
22.3	Derrida plots of the elementary rules	278
22.4	Derrida plots of $v2k5$ totalistic rules	278
22.5	Rule cluster examples	279
23.1	Basins of attraction — the idea	281
23.2	Point attractors (period=1)	282
23.3	The pre-image fan of a single pre-image of a point attractor	282
23.4	Attractors with period 2	282
23.5	Attractors with periods $\geq 3$	282
23.6	Subtrees from a root state	284
24.1	The state-space matrix, $n=6$	290
24.2	The state-space matrix, $n=12,13$	291
24.3	State-space matrix, further examples	292
24.4	In-degree histogram of a basin of attraction field, $n=18$	293
24.5	Rescaling the in-degree histogram of a subtree	294
24.6	in-degree histogram log-log	296
24.7	Density classification in emergent computation	297
24.8	A subtree showing exceptions to density classification	297
24.9	Basin of attraction field of a density classification rule	298
24.10	A view of rule-space	299
24.11	The attractor basin window while drawing graphs	301
24.12	Plotting $G$ -density against a range of $n$	301
24.13	$G$ -density against both $\lambda_{ratio}$ and $Z$	302
24.14	$\lambda_{ratio}$ - $Z$ plot	303
24.15	$Z_{left}$ - $Z_{right}$ graph	304
24.16	“Backwards” space-time patterns	307
25.1	Layout preview, single basin	310
25.2	Layout preview, basin field	310
25.3	The layout preview	311
25.4	Resetting transient parameters	312
25.5	The basin of attraction field for a range of network size	314
25.6	Amending the layout of a basin of attraction field on-the-fly	316
26.1	1d CA Basin of attraction field with null boundary conditions	319
26.2	1d RBN basin of attraction field with null boundary conditions	319
26.3	Compression of equivalent dynamics	320
26.4	Compression deactivated	320
26.5	CA uniform state attractors	321
26.6	Suppressing equivalent subtrees	322
26.7	Alternative node display	323
26.8	Symmetric states only	324
26.9	Just leaf-states or exclude leaf-states	325
26.10	Bit pattern colors of a uniform state’s pre-images	325
26.11	Highlighting all non-equivalent attractor states	326

26.12	Highlighting all attractor states in a RBN . . . . .	326
26.13	Changing the orientation of attractor basins . . . . .	328
26.14	Decreasing the pre-image fan-angle of a subtree . . . . .	329
26.15	Increasing the pre-image fan-angle of a basin of attraction . . . . .	329
26.16	Alternative edge start colors . . . . .	330
27.1	A basin of attraction field of a CA illustrating data . . . . .	333
27.2	A subtree for a CA, $n=150$ . . . . .	336
27.3	subtree=basin . . . . .	337
27.4	Prototype subtrees from a uniform state, CA, $n = 14$ . . . . .	338
27.5	A single CA basin rule 110 . . . . .	341
27.6	Detail of basin rule 110 . . . . .	343
27.7	Listing states of a basin of attraction field . . . . .	344
28.1	Mutation by one wire move . . . . .	349
28.2	Single basin mutation by one wire-move . . . . .	350
28.3	32 one-bit mutant basins of attraction . . . . .	351
28.4	Mutation of field by one bit-flip . . . . .	352
28.5	Single basin mutation by one value-flip . . . . .	353
29.1	Basin of attraction — limiting backward steps . . . . .	359
29.2	Game-of-Life glider-gun attractor . . . . .	359
29.3	X-rule glider-gun attractors, GGa and GGb . . . . .	360
29.4	Partial pre-image stack of a 1d local CA . . . . .	361
29.5	Partial pre-image stack of a 1d nonlocal CA . . . . .	363
29.6	The final partial pre-image histogram, not reordered . . . . .	364
29.7	CA, DDN/RBN, and random maps . . . . .	365
29.8	Random map graphs with different Hamming bias . . . . .	368
29.9	Basin of attraction field with sequential updating . . . . .	371
29.10	The rule window for sequential updating . . . . .	373
29.11	Neutral subtree . . . . .	374
29.12	The neutral field . . . . .	375
30.1	The basin of attraction field progress bar. . . . .	378
30.2	Drawing a basin of attraction field . . . . .	379
31.1	1d space-time pattern, colors by value and neighborhood . . . . .	388
31.2	2d space-time pattern, colors by value and neighborhood . . . . .	389
31.3	3d space-time pattern, colors by value and neighborhood . . . . .	389
31.4	State space matrix . . . . .	390
31.5	Return map . . . . .	390
31.6	Probabilistic updating . . . . .	394
31.7	Sequential and partial order . . . . .	396
31.8	Pattern Density and Input-Entropy plots . . . . .	398
31.9	The difference in the dynamics between two 1d CA . . . . .	400
31.10	A duplicated 2d network . . . . .	401
31.11	A duplicated 2d seed . . . . .	401
31.12	2d “damage” . . . . .	402
31.13	2d automatic statistics on damage . . . . .	403
31.14	The damage histogram showing actual damage . . . . .	405
31.15	The damage histogram showing damage bins . . . . .	405
31.16	The attractor histogram for density classification . . . . .	408



31.17	The attractor frequency histogram	409
31.18	The attractor histogram sorted by different measures	410
31.19	Transient data	414
31.20	The attractor histogram jump-graph	415
31.21	The skeleton histogram	417
31.22	Examples of frozen skeletons	417
31.23	The skeleton frequency histogram	419
32.1	On-the-fly key index for space-time patterns	423
32.2	Setting a random k-rcode on-the-fly	430
32.3	1d isotropic rules	431
32.4	Mutate/restore on-the-fly by a random bit/value flip	432
32.5	Info when changing $Z$ on-the-fly	433
32.6	Examples of $v2k5$ complex space-time patterns with interacting gliders	434
32.7	Space-time 2d snapshots of complex kcode	435
32.8	Loading complex rules on-the-fly	436
32.9	Info when loading a complex rule on-the-fly	436
32.10	Randomizing wiring by stages	437
32.11	A positive and negative singleton seed	439
32.12	Slanting 1d space-time patterns	441
32.13	Hex or square 2d lattice	441
32.14	Divisions between cells	441
32.15	The cell outline in network-graph space-time patterns	442
32.16	Skipping time-steps	444
32.17	Time-step spacing in scrolling network-graph STP	444
32.18	A 3d network shown in 2d	445
32.19	3d cells shown as balls or parallelograms	446
32.20	2d+time, 2d diagonal scrolling — toggle “balls”	446
32.21	Gilders with time-trails	447
32.22	Top-right 5-way space-time pattern info	448
32.23	Alternative presentations, 2d space-time patterns	449
32.24	2d CA frequency bins	451
32.25	frequency bins set by hand	451
32.26	Examples of filtering space-time patterns	452
32.27	Filtered lookup-frequency histogram	452
32.28	Filtering a complicated background domain	453
32.29	Filtering showing discontinuities within a chaotic domain	453
32.30	Input frequency histogram presentation	454
32.31	The input frequency histogram in 3d	455
32.32	Entropy/density scatter plot	456
32.33	Return map by density	458
32.34	Return map by density — randomly wiring a complex rule	458
32.35	Directly scanned 1d space-time patterns for vector PostScript	470
32.36	PostScript 1d space-time patterns, dots and divisions	470
32.37	1d CA space-time patterns in network-graph layout	472
32.38	2d CA space-time patterns within the network-graph	473
32.39	Network-graph vector PostScript	474
33.1	A 2d histogram of an entropy-variability scatter plot	475
33.2	Input-entropy variability	476
33.3	1d $v8k5$ tcode sample	477

33.4	2d histogram - other presentations	477
33.5	1d <i>v8k3</i> kcode sample	478
33.6	Running a test	481
33.7	Creating an automatic sample	483
33.8	Loading, sorting and displaying a sample	484
33.9	The scatter plot color scheme	486
33.10	rule sample scatter plot	487
33.11	2d frequency histogram of the rule sample scatter plot	487
33.12	scatter plot grid and 2d histogram at a lower resolution	488
33.13	rule sample scatter plot, $Z$ -parameter and $\lambda$ -ratio	490
33.14	On-the-fly info when scanning sample space-time patterns	491
33.15	2d <i>v4k7</i> kcode sample	492
33.16	Scanning a 2d CA in blocks of time-steps scrolling diagonally	493
33.17	Listing the rule sample	494
33.18	Listing by plot coordinates or probing with the mouse	496
34.1	The basin of attraction field of a RBN, before learning	498
34.2	The basin of attraction field of a RBN after learning	499
34.3	The RBN networks before and after learning	499
34.4	Nodes other than target and pre-image suppressed	500
34.5	Selecting parity	503
34.6	Highlighting states in rule 110	508
34.7	Highlighting states in a $v=4$ CA	509
35.1	Listing files	516

## List of Tables

7.1	$k_{Lim}$ for different value-range $v$	73
7.2	Basin of attraction field: $n_{Lim}$ for different value-range $v$	73
8.1	Times for increasing $k$ and $n$ for $v=2$ , basin of attraction fields, 1d CA.	81
8.2	Times for increasing $v$ and $n$ for $k=3$ , basin of attraction fields, 1d CA.	81
8.3	Times for increasing $k$ and $n$ for $v=2$ , basin of attraction fields, RBN.	82
8.4	Times for increasing $v$ and $n$ for $k=3$ , basin of attraction fields, DDN.	82
9.1	Effective $k=0$ neutral rules	84
13.1	The size of rule-tables $k = 1$ to 13	112
13.2	The size of rule-tables	114
13.3	kcode size: for $v$ and safe $k_{Lim}$	116
13.4	The size of kcode-tables	116
13.5	kcode $v=3$ matrix	117
13.6	tcode size: for $v$ and $k_{Lim}$	117

14.1	Post function list $v2k3$	133
14.2	Post function list $v2k5$	133
14.3	Post function list (multi-value) $v3k5$	134
16.1	The default rule value-bias	146
16.2	Rules in decimal — rule-table size limitations	154
16.3	Immediate rule data in the terminal	163
16.4	Rule details in the terminal	164
16.5	Horizontal and matrix layout options for kcode in the terminal	165
16.6	Swapped kcode — some terminal layouts	165
20.1	The RBN used in the network-graph and jump-graph examples	222
20.2	The adjacency-matrix of the network-graph	241
20.3	The jump-table	242
21.1	2d and 3d seeds in the terminal	246
21.2	The default seed value-bias	250
21.3	Seeds in decimal — network size limitations	263
24.1	Canalyzing, $\lambda_{ratio}$ and $P$ data, $k=3$	305
24.2	Canalyzing, $\lambda_{ratio}$ and $P$ data, $k=4$	305
26.1	Alternative presentations bit/value nodes	327
29.1	Maximum network size for the exhaustive testing algorithm	365
29.2	Printing exhaustive pairs $v2$	366
29.3	Printing exhaustive pairs $v4$	366

# Chapter 1

## Overview

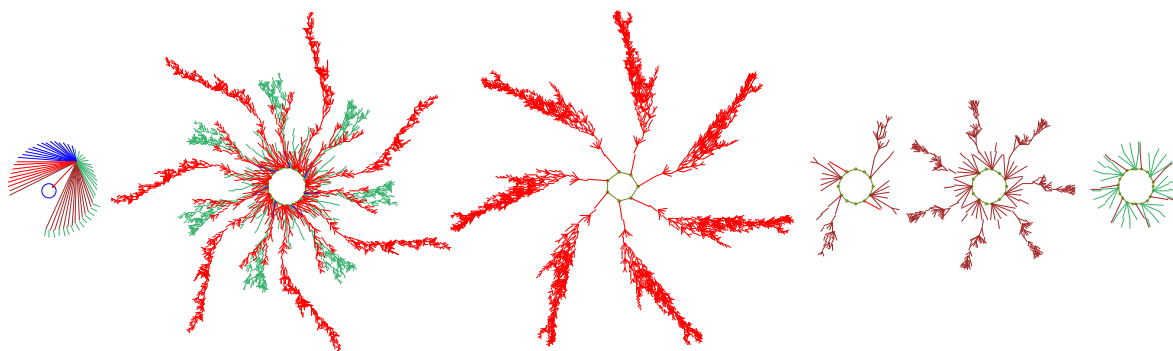


Figure 1.1: The basin of attraction field of a Cellular Automaton,  $v=2$ ,  $k=3$ ,  $n=14$ . There are in fact 15 basins, but equivalent basins have been suppressed leaving just the 6 prototypes. State nodes have been omitted. This is one of the computationally universal “elementary” rules, 193 (equivalent to the famous rule 110 [28])

---

### 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. The term “attractor basin” refers to any type of state transition graph: a basin of attraction field, single basin, or subtree.

Figure 1.2 gives a glimpse of the main themes in DDLab, and also the broad, slippery, and overlapping categories of the systems available, listed below (see also figure 29.7),

- CA: Cellular Automata: a local neighborhood of  $k$  inputs (1d, 2d, 3d, with periodic or null boundary conditions), and one rule (but possibly a mix of rules to extend the definition).
- RBN: Random Boolean Networks: non-local or (possibly biased) random wiring of  $k$  inputs (but possibly with mixed- $k$ ,) and a mix of rules (but traditionally just one rule).

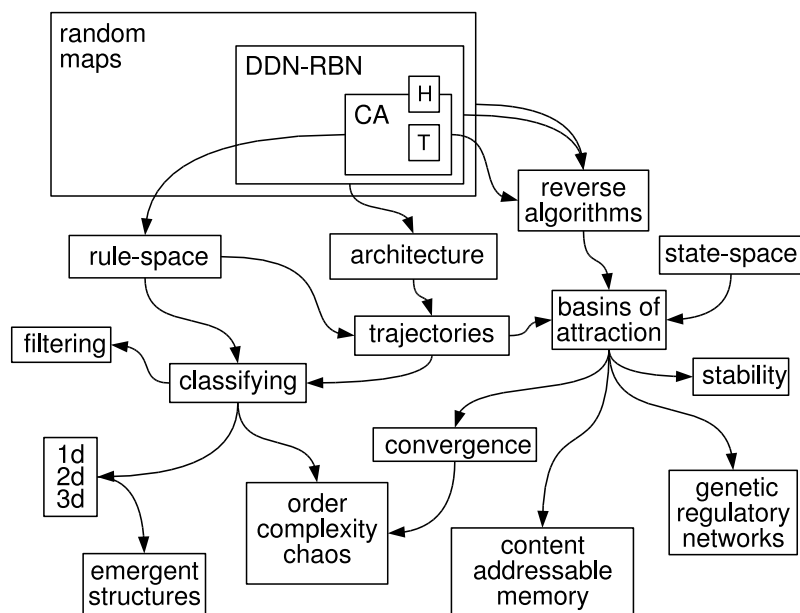


Figure 1.2: The various themes, methods, functions and applications in DDLab, loosely connected. *Top Left:* The expanding hierarchy of networks: CA  $\rightarrow$  RBN/DDN  $\rightarrow$  within the super-set of random maps (directed graphs with out-degree one), imposing decreasing constraints on the 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 ( $k = n$ ), mixed rule CA and the rest are equivalent to random maps.

DDLab and its ideas have been used widely in research and education, and have been applied to study complexity, emergence and self-organization [31, 38, 45], in the behavior of bio-molecular networks such as neural [33, 34] and genetic [20, 25, 16, 36] networks, and in many other disparate areas — from physics and cosmology [11] to art [3] and creativity [2].

The results in publications [31] to [50] and in this book can be implemented with DDLab.

As well as scientific applications, DDLab’s imagery has featured in art exhibitions [39], as the light show at events, and has been applied for generative music [6, 7, 8].

---

## 1.2 Source code and platforms

The DDLab code is free software under the GNU General Public License (GPL) as published by the Free Software Foundation. Registration is subject to a modest fee, whereby the annoying UNREGISTERED banner can be easily removed (section 5.4).

Compiled versions of DDLab (32-bit unless specified as 64-bit) are maintained for UNIX based (Linux-like) operating systems, and also for DOS, as follows:

- Linux/PC: compiled in Ubuntu 6.06 and Ubuntu 11.04 (64-bit).
- MAC OSX with X11: compiled in PowerPC G4 10.4.11, MacBook Pro 10.5.8, and Mountain Lion 10.8.2.
- CygwinX/Windows/PC: this is a Linux environment running in MSWindows98 and above. The default packages plus `xorg-x11-devel`, `xorg-x11-fnts`, must be installed. The CygwinX version is preferable to the DOS version below.
- DOS/PC: compiled with WatcomC 11. There is no native MSWindows version of DDLab at present.

The following compiled versions from 2005 are available but no longer maintained, though up-to-date versions could be compiled from source:

- UNIX/XWindows/Sun: compiled in SunOS 5.8. The libraries `libX11.so.6.1` and `libsunmath.so.1` need to be accessible.
- Irix/SGI: compiled for IRIX 6.5.27 with MIPSpro C 7.4.3m and `-n32`.

The manual covers all maintained 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/MAC/CygwinX/UNIX/Irix 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, downloads, installation, documentation, registration, platforms, source code, compiling, updates, examples of output, applications, and related publications, consult one of the DDLab web sites below:

<http://www.ddlab.org>

<http://users.sussex.ac.uk/~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 automata, elements<sup>1</sup> 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.

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.

<sup>1</sup>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.

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

- SEED, FIELD or TFO mode: An initial choice to establish the type of system (section 6.1). SEED-mode requires a seed (an initial state) for running forward or seeding a single basin of attraction. FIELD-mode is for a basin of attraction field which does not require an external seed to be set — seeds are set automatically. TFO-mode (Totalistic Forwards-Only) restricts DDLab to running forward only from a seed according to just totalistic rules. These initial choices effect maximum safe sizes — of the network  $n$ , and its connectivity  $k$ . An additional choice, “exLimits” allows these safe sizes to be exceeded.
- The “value-range”,  $v$ : The range of values (cell-states) available to a cell, currently from 2 to 8 (chapter 7). In other words, the number internal states, or colors, or letters in a cell’s “alphabet”. Prior 2003 DDLab was limited to binary or Boolean logic — just 2 values (0,1).
- The system size,  $n$ : The number of network elements. Maximum  $n$  is much less in FIELD-mode than in SEED/TFO-modes, but the new limits (section 8.3) have increased since the 2011 edition.
- 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 (chapter 10). 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. Maximum  $k$  currently supported in DDLab ( $k_{Lim}$ ) depends on the value-range  $v$ , and has increased since the 2011 edition (table 7.1).
- The “wiring scheme”: defining the location of the output terminals of each cell’s input wires — its “neighborhood” (e.g. section 17). 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, or to make scale-free networks. 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 (e.g. section 14). Each element applies a rule to its inputs to compute its output. Usually this is made into a lookup-table, the “rule-table”, listing the outputs of all possible input patterns (rcode), or some subset of input patterns (e.g. kcode or tcode). 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 outer-totalistic rules where the rule to be applied depends on a cell’s current value, thus requiring  $v$  separate rules. Outer totalistic rules are used to implement reaction-diffusion, the game-of-Life, and other Life-like rules.

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

Acronym glossary: (chapter 37 provides a detailed 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 [19], 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 for space-time patterns, and *global* dynamics, running the network backwards for attractor basins — the idea is explained in figure 23.1.

Running “backwards” generates multiple predecessors rather than a trajectory of unique successors. This procedure reconstructs the branching subtree 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 subtree. subtrees, 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.

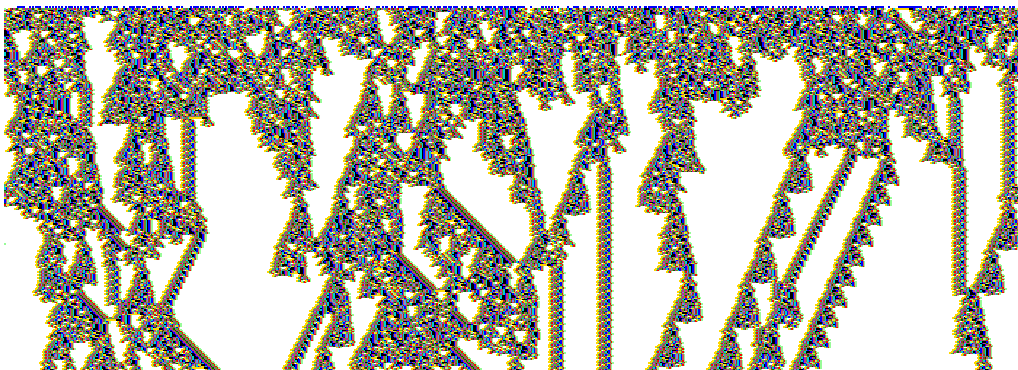


Figure 1.3: The space-time pattern of a 1d complex CA with interacting gliders. 308 time-steps from a random initial state. Value-range  $v=2$ , neighborhood size  $k=7$ , system size  $n=700$ , rcode(hex)=3b46 9c0e e4f7 fa96 f93b 4d32 b09e d0e0. Cells are colored according to neighborhood look-up instead of the value. Space is across and time down the page. The basin of attraction field for this rule ( $n=16$ ) is shown figure 1.4.



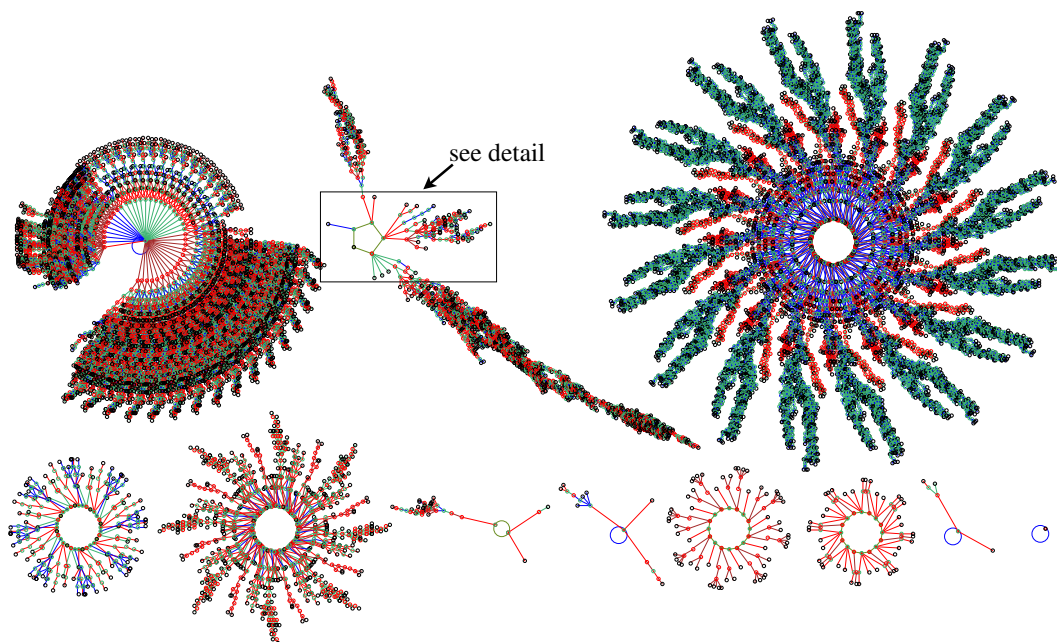


Figure 1.4: The basin of attraction field of a complex  $v2k7$  CA,  $n=16$ , defined in figure 1.3, which shows its space-time patterns. The  $2^{16} = 65536$  states in state space are connected into 89 basins of attraction. Only the 11 non-equivalent basins are shown, with symmetries characteristic of CA [31]. 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,  $\lambda_{ratio}=0.938$ ,  $Z=0.578$ .

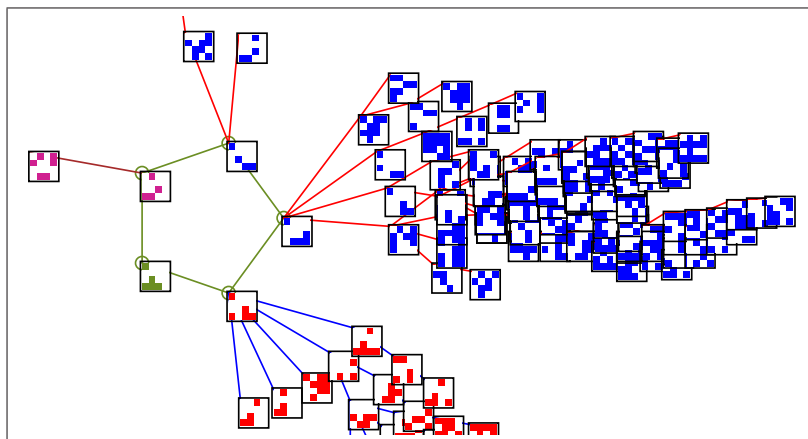


Figure 1.5: A detail of the 2nd basin of attraction in figure 1.4. The states are shown as  $4 \times 4$  bit patterns.

### 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 (e.g. chapter 13). The rule-tables of totalistic rules (tcode and kcode) are much smaller than full rule-tables (rcode), so larger neighborhoods are possible ( $k_{Lim}=25$ , 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 [32, 33]. Each basin is categorized by its attractor and each subtree 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 (chapter 34).

## 1.6 Size limits: network $n$ and neighborhood $k$

The size limits of networks  $n$  and neighborhoods  $k$  have increased since the 2011 edition, especially for 64-bit systems. The usual limits are indicated below, but can be increased with a new option for extra limits — “exLimits”. However, high values of  $n$  or  $k$  may be impractical because of time and memory constraints.

### 1.6.1 Network size limits

Whereas large networks may be run forward to look at space-time patterns (SEED/TFO-modes), or backward to look at subtrees (SEED-mode), the system size  $n$  is limited when generating the entire basin of attraction field (FIELD-mode), given that state-space  $S$  grows exponentially with  $n$  ( $S = v^n$ , where  $v$  is the value-range). DDLab’s upper limits,  $n_{Lim}$ , (including “exLimits” — extra limits) in FIELD-mode are listed in table 7.2. For binary systems the basic  $n_{Lim}$  is 31 (with exLimits active 35).

In SEED/TFO-modes the basic limit of  $n$  is 8388607 (with exLimits active 4294967295). For 2d  $i, j$  or 3d  $i, j, h$  any sub-multiples of  $n$  apply — for a square  $2896 \times 2896$ , for a cube  $203 \times 203 \times 203$ . In practice much smaller sizes are appropriate for single basins and subtrees<sup>2</sup>. Large size may impose unacceptable time, memory or display constraints. For RBN and DDN, running backwards generally imposes a greater computational load than for CA.

<sup>2</sup>Networks with disordered (chaotic) dynamics, which have low in-degree or branchiness in their subtrees such as CA chain-rules [47], 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 [38].

## 1.6.2 Neighborhood size limits

The maximum neighborhood size  $k$ , depending on value-range  $v$ , is set out in table 7.1. For binary networks ( $v=2$ ) the upper limits of  $k$ ,  $k_{Lim}$ , is 27, which allows a  $3 \times 3 \times 3$  neighborhood in a 3d network. The (pseudo-)neighborhoods are predefined in chapter 10.

## 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.

Space-time patterns can be altered on-the-fly (chapter 32), including changes to rules, connections, current state, scale, and alternative presentations highlighting different properties. 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 (chapter 35), and most graphic output can be saved as vector PostScript files (chapter 36).

## 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:  $\lambda$ , 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 its variability — which allows ordered, complex and chaotic rules to be classified automatically [38].
- 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, and information about the GPL license, copyright and registration.
- Chapter 4 describes DDLab's graphical user interface and gives some “quick start” examples. It's 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 36 contain the detailed reference manual.

For further background on the attractor basins of CA, RBN, DDN, and their implications, see publications [31]-[50], which are available at:

<http://uncomp.uwe.ac.uk/wuensche/publications.html>

---

# 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 choice 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 rule-tables and all attractor functions (prompts for these will not be displayed), and allows larger neighborhoods, ( $k_{Lim}=27$ , 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 there is a further choice, to set either, FIELD-mode — for the basin of attraction field, which does not require a seed (initial state), or SEED-mode — for anything else (which *does* require a seed) — space-time patterns, a single basin of attraction, or a subtree. This choice can be amended in at a later stage, in section 30.4.

### 2.2.3 Exceeding normal limits of $n$ and $k$

A new initial option since the 2011 edition “exLimits” allows exceeding the usual limits of system size  $n$  and neighborhood size  $k$ , but with a warning that RAM may be exceeded (section 6.2.4).

---

## 2.3 Setting the value-range



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 color keys.

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,  $k_{Lim}$  will decrease (section 2.5), depending on whether TFO-mode was set in section 2.2.1. The selection of the value-range can only be made at this early stage in the program.

---

## 2.4 Setting the network size

The network size  $n$  for 1d is set early on in the prompt sequence, but this is superseded 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 32.8.6), in general the network size can only be changed at these early prompts.

For space-time patterns, the upper limit of network size  $n_{Lim}=8388607$  (with exLimits active 4294967295). For 2d  $i, j$  or 3d  $i, j, h$  any sub-multiples of  $n$  apply — this allows a square  $2896 \times 2896$ , and a cube  $203 \times 203 \times 203$ . This limit also applies for single basins and subtrees, though in practice much smaller sizes are appropriate, except when generating subtrees for maximally chaotic CA chain-rules (section 16.11).

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). For example, for  $v=8$  and unconstrained DDLab,  $k_{Lim}=4$  to handle the large lookup-table whereas in TFO-mode  $k_{Lim}=11$ .  $k_{Lim}$  for increasing value-range  $v$ , for both cases, is set out in section 7.2.

$k$  can be homogeneous, or there can be a mix of  $k$ -values in the network. The  $k$ -mix may be set and modified in a variety 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).

## 2.6 Wiring

The network's wiring scheme (i.e. its connections) has predefined neighborhood templates for local CA (for 1d, 2d and 3d) for neighborhood size,  $k=1$  to  $k_{Lim}$  (chapter 10). The 3d templates define a cuboid lattice with periodic boundary conditions. In 2d, the templates define a toroidal lattice which can be either square, hexagonal or new hex/triangular lattice for  $k3$  or  $k4$  (figure 2.2). The square lattice includes the 5-cell von Neumann neighborhood and the 9-cell Moore neighborhood. Wiring can also be set at random (nonlocal wiring), with a wide variety of constraints and biases, or by hand (chapter 12). The predefined templates 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, which can may be saved/loaded (chapter 19).

Nonlocal wiring can be constrained in various ways (section 12.5) including confinement within a local zone 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, and 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).

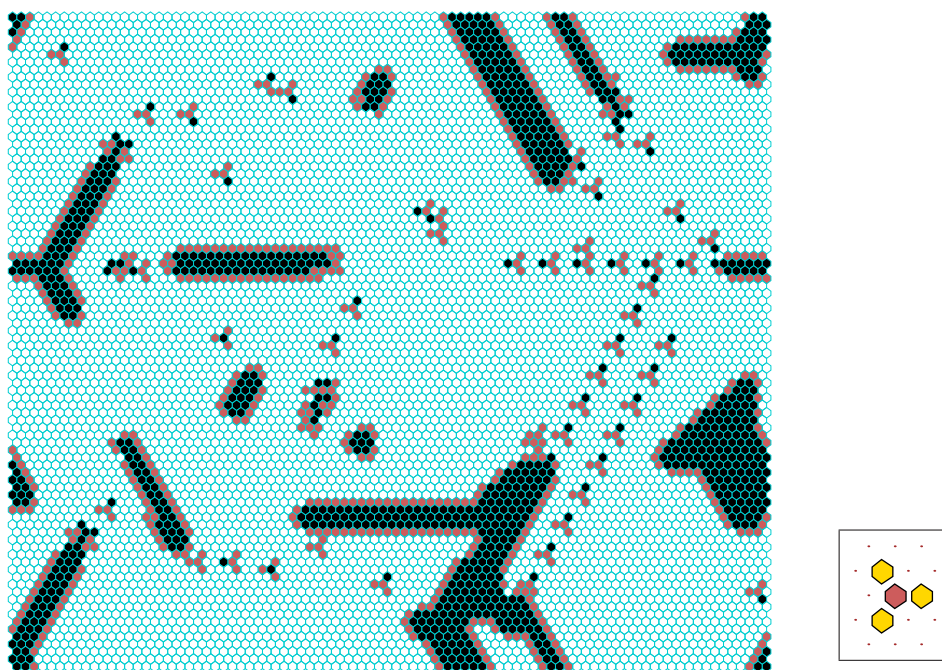


Figure 2.2: New 2d hex/triangular neighborhood templates are available for  $k3$  and  $k4$  neighborhoods, which permit investigating the dynamics on these simpler lattices, with many instances of complexity. The example shown is a snapshot of  $v3k4$  kcode 2a945900 from a random initial state ( $88 \times 88$ ) — growing and stable structures emerge which act as glider-guns. The inset shows the  $k4$  hex/triangular neighborhood template from figure 10.2.



---

## 2.7 Null boundary conditions

By default, a network’s neighborhood (or pseudo-neighborhood) is assigned with periodic boundary conditions (PBC), where lattice edges (and wiring inputs) wrap around to their opposite edges. This makes a ring in 1d, a torus in 2d, and a 3-torus in 3d.

New options (sections 26.1, 31.3, 32.7.2) allow null boundary conditions (NBC), where inputs beyond the network’s edges are held at a constant value of zero. NBC are of interest in pattern recognition, and other applications where the system is grounded or quenched, or bounded by an edge, skin or membrane. As for PBC, NBC and their dynamics are interesting as mathematical/dynamical systems in their own right.

All DDLab functions and options for computation, display and analysis can now also be applied to NBC systems, multi-value as well as binary. This includes CA in various dimensions, but also random Boolean networks (RBN) and discrete dynamical networks (DDN), because irrespective of the wiring scheme, for NBC any part of the pseudo-neighborhood that extends beyond the network’s edges is made to take zero as its input.

NBC space-time patterns (running forward) apply to 1d, 2d and 3d networks (section 31.3) – NBC and PBC can be toggled one on-the-fly (section 32.7.2). All (forward) functions and methods apply — such as the look-up entropy, filtering, damage, and attractor histograms.

NBC basins of attraction (running backward) apply to 1d networks by means of three completely different reverse algorithms, where the original PBC algorithms have been modified for NBC: (1) for CA, (2) for RBN/DDN as well as CA, and (3) the exhaustive reverse algorithm for any of the above, thus providing a reality check of results. All (backward) methods and functions for basin of attraction fields, single basins and subtrees apply (section 26.1).

---

## 2.8 Rules

The most general update logic or rule is expressed as a full rule-table (lookup-table), referred to as “rcode”, but there are useful subsets of the general case, two types of totalistic rules, “kcode” and “tcode” (chapter 13). 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. 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 implementation of reaction-diffusion rules or excitable media [14].

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 rcode. Transformations and mutations would then apply to just the restricted rule-table, kcode or tcode.

If DDLab remains unconstrained, tcode and kcode (but not outer-totalistic rules) can still be selected, but they will be transformed into the full rcode rule-table, which allows attractor basins. Transformations and mutations will apply to the rcode. Within rcode there are also subsets of rules that can be automatically selected at random, including isotropic rules, majority rules, maximally chaotic chain-rules, Altenberg rules, and “game-of-Life” rules.

A network may have one homogeneous rule as for CA (chapter 16), or a rulemix as for RBN and DDN (chapter 14). The rulemix 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. A rule scheme can be set and amended just for a predefined sub-network within the network, and can be saved/loaded (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  $\lambda$  and  $Z$ , and the frequency of canalizing inputs in a network can be set to any arbitrary level (chapter 15).

## 2.9 Initial network state, seed



Figure 2.3: Drawing a 2d seed “portrait”,  $88 \times 88$ ,  $v=8$ , with the mouse and keyboard, shown top-left. The seed was then transformed by applying a CA majority rule ( $v8k4$ ) for three time-steps.

An initial network state (a seed, chapter 21) 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/RBN/DDN, most

states in state-space have no predecessors (they are the leaves of a subtree, “garden-of-Eden” states), so from a random seed it is usually necessary to run forwards by a few steps to penetrate the subtree before running backwards — this option provided is (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 variety of methods for defining the seed (chapter 21), in decimal or hex, as a bit pattern in 1d, 2d or 3d, or at random with various constraints or biases. The bit pattern method is a mini paint program, using the mouse and keyboard to draw colors. Figure 2.3 shows the method applied to draw a portrait, which is then transformed by applying a CA rule for three time-steps. When loading, a seed-file can differ in value-range, size, and dimension from the base network (section 21.7).

## 2.10 Networks of sub-networks

It is 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 (section 19.4). 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 different from the base network, provided the base network is set up appropriately to accommodate the sub-network. For example, to load an DDN into a CA, the CA may need to be set up as if it were an DDN. To load a mixed- $k$  sub-network into a 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.

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 31.6).

## 2.11 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”.

### 2.11.1 Space-time patterns

*chapters 31 and 32*

A cell in a space-time pattern is colored according to its value, or alternatively according to a predefined color depending on its neighborhood at the previous time step, the entry in the rule-table that determined the cell’s value (figure 2.4). Space-time patterns can be filtered to suppress cells that updated according to the most frequently occurring neighborhoods, and 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 — these functions are able to expose “gliders” and other structures (section 32.11). The emergence of such frozen elements is associated with “canalyzing inputs”, and underlies Kauffman’s RBN model of gene regulatory networks [20, 16].

1d space-time patterns are usually 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

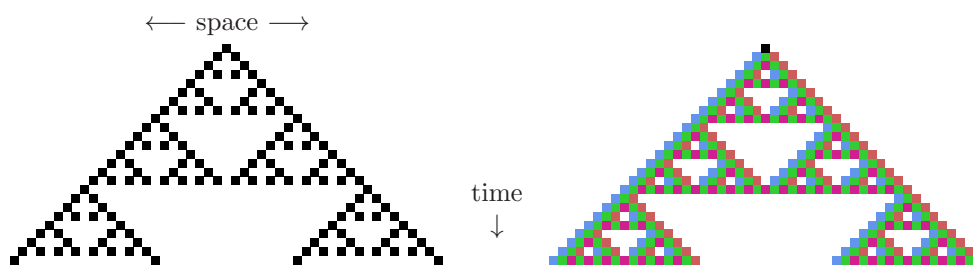


Figure 2.4: Space-time patterns of a 1d CA,  $v2k3$ ,  $n=51$ , rcode (dec)90. 24 time-steps from an initial state with a single central 1. Two alternative presentations are shown. *Left*: cells by value. *Right*: cells colored according to their look-up neighborhood.

dimension ( $2d+time$ ) where successive time-steps scroll 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 (section 32.19). For example a 1d space-time pattern can be shown in a circular layout which can also be scrolled (i.e. figure 4.9).

### 2.11.2 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 subtree (also from a seed). Because a random seed is likely to be a garden-of-Eden state, to generate subtrees an option is offered to run the network forward a given number of steps to a new seed before running backward. This guarantees a subtree 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). Local 1d and 2d CA produce attractor basins where subtrees 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 simultaneously 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 (figure 4.17, section 24.8).

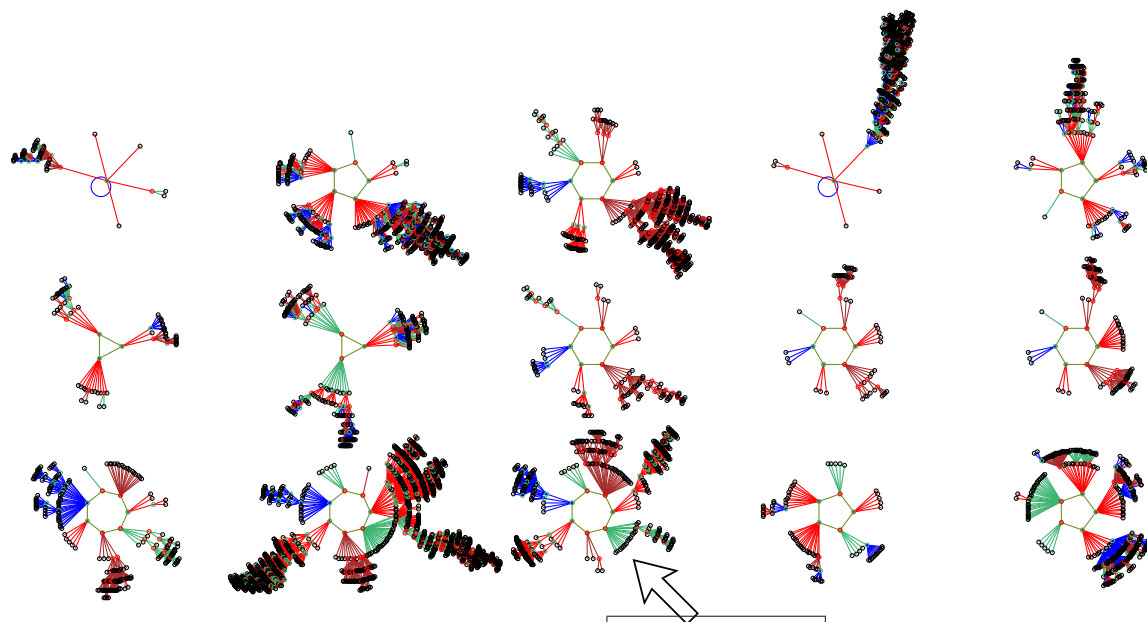


Figure 2.5: The basin of attraction field of a random Boolean network,  $v2k3$ ,  $n=13$ , (also shown in the jump-graph, Figure 2.3). The  $2^{13} = 8192$  states in state space are organized into 15 basins, with attractor periods ranging between 1 and 7. The number of states in each basin is: 68, 984, 784, 1300, 264, 76, 316, 120, 64, 120, 256, 2724, 604, 84, 428. figure 2.6 shows the arrowed basin in more detail. *Right:* the network's architecture, its wiring/rule scheme (n13RBN.wrs — section 3.6).

this basin shown  
in more detail in  
figure 2.6

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

### 2.11.3 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).

## 2.12 Graphics

In Linux-like versions, the DDLab screen starts up at  $925 \times 694$  or a smaller size automatically set to comfortably fit on the monitor. This can be resized, moved and iconized in the usual way. In DOS the graphics will start up at a resolution of  $640 \times 480$  (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.

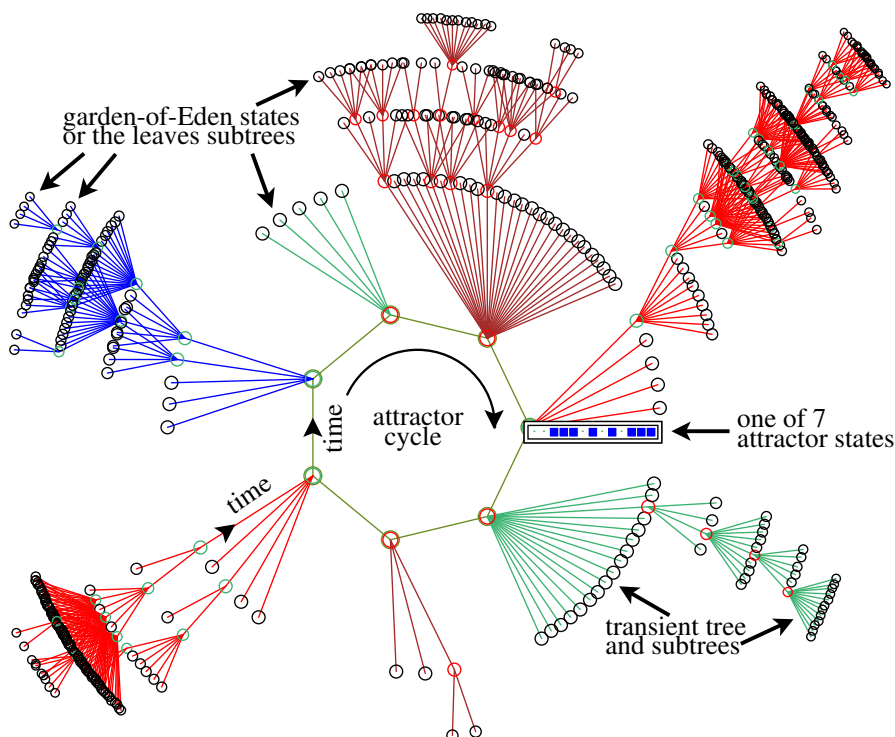


Figure 2.6: A basin of attraction (one of 15) of the random Boolean network,  $v2k3$ ,  $n=13$ , shown in figure 2.5. The basin links 604 states, of which 523 are garden-of-Eden states. The attractor period = 7, and one of the attractor states is shown in detail as a bit pattern. The direction of time is inwards from garden-of-Eden states to the attractor, then clock-wise.

## 2.13 Filing and Printing

DDLab allows filing a wide range of file types, including network parameters, data, the screen image and vector PostScript files. (sections 19, 35). For compatibility with DOS, filenames follow the DOS format, so a filename 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 extension (or a default filename can be accepted), the extension is added automatically to identify the file type.

### 2.13.1 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).

### 2.13.2 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 automatically generated and saved (chapter 33), This allows sorted samples of CA rules to be created, discriminating between order, complexity and chaos (chapter 33), and complex rules, those featuring “gliders” or other large scale emergent structures, to be collected automatically; some collections/samples of 1d and 2d CA rules are provided with DDLab (sections 32.6.1, 3.6.4).

---

## 2.14 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 (chapter 36). Vector graphics is preferable for publication quality images. The methods work in both Linux-like systems and DOS. Images saved as vector PostScript files can be printed in GhostView, or converted to .pdf files and printed in Adobe (acroread).

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

---

## 2.15 Bitmap images

The screen image can be saved and loaded using an efficient compressed format only applicable within DDLab (section 5.5). Alternatively, in Linux-like systems, a program such as XView can be used to grab the DDLab screen 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.

---

## 2.16 Printing the screen image

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 (section 5.6). Alternatively use XView to grab the bitmap image, in various formats.

---

## 2.17 Mutations

A wide variety of network “mutations”, as well as changes in presentation, can be made, many on-the-fly, while running forward for space-time patterns, or backward for attractor basins.

### 2.17.1 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 32.6.1).

## 2.17.2 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).

## 2.18 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 graphically.

### 2.18.1 Behavior parameters

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

- The  $\lambda$  parameter (sections 14.1.2, 16.3.1) and equivalent  $P$  parameter.
- The  $Z$ -parameter [31, 38] (section 24.9).
- The (weighted) average  $\lambda$  and  $Z$  for a mixed rule network (section 17.9.2).
- The frequency of canalizing “genes” and inputs — for a rulemix network (chapter 15), for single rules (section 18.6).
- Post-function data (section 14.12).

$\lambda$ ,  $Z$ , and canalization can be set to any arbitrary level.

### 2.18.2 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, outputs, or both (all connections), in the network (section 17.9.13).
- The recursive inputs/outputs to/from a network element, whether direct or indirect, showing the “degrees of separation” between elements (sections 17.6.6, 17.6.7).
- The network-graph (section 20.2), where the wiring is analyzed in two ways, as an adjacency matrix (figure 20.2): a matrix showing links between cells, and as a network-graph (figure 20.1) a graph with (weighed) vertices and edges. The network-graph 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 (analogous to the jump-graph in section 2.18.4).

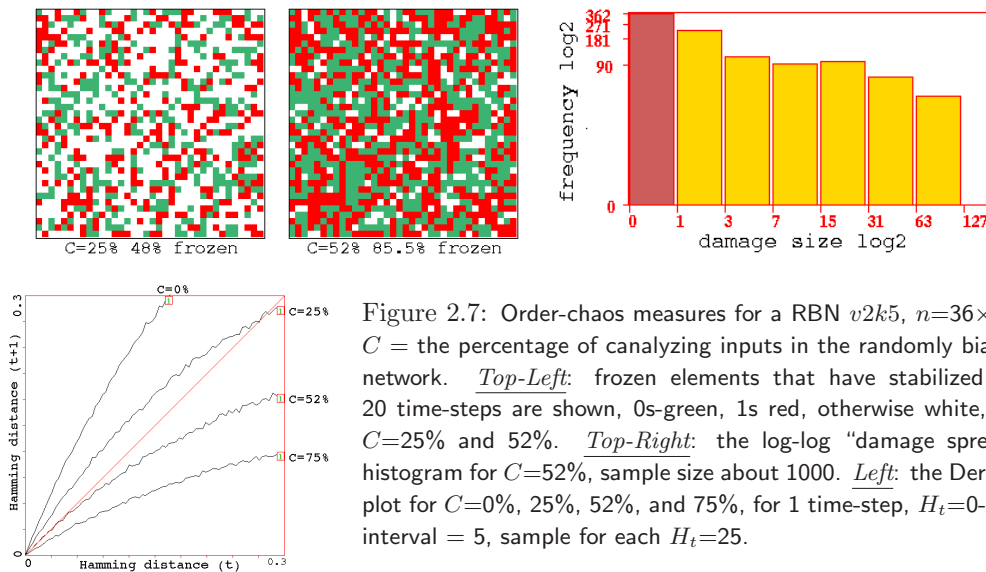


Figure 2.7: Order-chaos measures for a RBN  $v2k5$ ,  $n=36 \times 36$ .  $C$  = the percentage of canalizing inputs in the randomly biased network. *Top-Left*: frozen elements that have stabilized for 20 time-steps are shown, 0s-green, 1s red, otherwise white, for  $C=25\%$  and  $52\%$ . *Top-Right*: 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  $H_t=25$ .

### 2.18.3 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 (figure 32.30), which can be toggled between 2d and 3d to include a time dimension (figure 32.31).
- The entropy of the lookup frequency over time (section 32.12.4).
- The variability of the entropy, min-max or standard deviation (section 33.1), and the entropy/density scatter plot where complex rules have their own distinctive signatures (figure 32.32).
- A plot of mean entropy against entropy variability for large samples of CA rules, which allows ordered, complex and chaotic rules to be classified automatically (chapter 33), also shown as a 2d frequency histogram (figure 33.1).
- The pattern density in a moving window of time-steps (section 31.5).
- “Frozen” options allow visualizing the activity/stability of network elements (section 32.11) — the fraction of “genes” unchanged for  $x$  generations (figure 2.7 *Top-Left*), or that fall into preset “frequency bins” (figure 32.22).
- The damage spread, or pattern difference, between two networks differing by 1 bit or value (section 31.6). A histogram of damage distribution can be generated for a sample of initial state pairs (figures 2.7 *Top-Right* and 31.13).
- 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, figure 2.7 *Left*).
- A scatter plot of successive iterations in a 2d phase plane, the “return map by value”, showing fractal structure, especially for chaotic rules (section 31.2.2.2).



- A scatter plot of successive iterations, the “return map by density”, This can be interesting when the densities follow a periodic oscillation, which occurs for a network with random wiring but one complex glider rule (section 32.12.8).

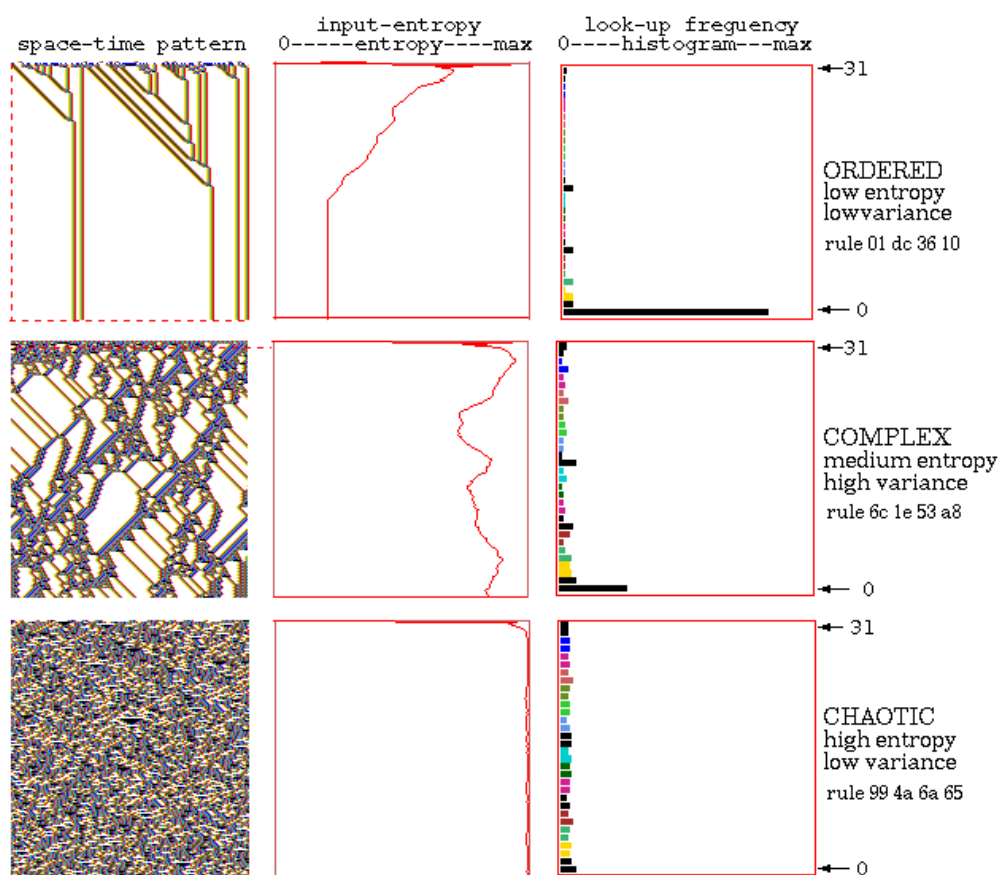


Figure 2.8: Typical 1d CA space-time patterns showing ordered, complex and chaotic dynamics,  $n=150$ ,  $v2k5$  rcodes shown in hex. Alongside each space-time pattern is a plot of the input-entropy, where only complex dynamics exhibits high variability, caused by glider collisions.

#### 2.18.4 Measures on global dynamics

Measures on global dynamics, i.e. attractor basins — the basin of attraction field, single basins and subtrees, are available as follows,

- 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 showing the frequency of arriving at different attractors from a sample of random initial states (figure 31.17) 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 measured statistically (section 31.7). A new feature analyses the transient data to record the set of unique transients states to each attractor, without duplication (figure 31.19). An attractor jump-graph can be constructed from this data (figure 31.7.8).

An analogous methods show the frequency of arriving at different partly frozen patterns, frozen attractors called “skeletons” (section 31.8).

- Garden-of-Eden density plotted against network size (figure 24.12), and against the  $\lambda$  and  $Z$  parameters (figure 24.13).
- A histogram of the in-degree frequency (section 24.6) in attractor basins (figure 24.4), in subtrees (figures 2.9, 24.5).
- The state-space matrix (section 24.5), a scatter-plot of the left half against the right half of each state bit/value string, using color to identify different basins, or attractor cycle states (figure 24.2).
- 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 20.2). The information is presented in two ways, as a jump-table (figure 20.3): a matrix showing the jumps between basins, and as a jump-graph: (figure 20.12) a graph with (weighed) vertices and edges. The jump-graph 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 (analogous to the network-graph in section 2.18.2).

## 2.19 Reverse algorithms

There are three different reverse algorithms for generating the pre-images of a network state, thus generate attractor basins.

- An algorithm for local 1d wiring [31] — 1d CA but rules can be heterogeneous.
- A general algorithm [32] for RBN, DDN, 2d or 3d CA, which also works for the above.
- An exhaustive algorithm that works for any of the above by creating a list of “exhaustive pairs” from forward dynamics. Alternatively, a random list of exhaustive pairs can be created to implement attractor basin of a “random map” (section 2.20).

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 local 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.

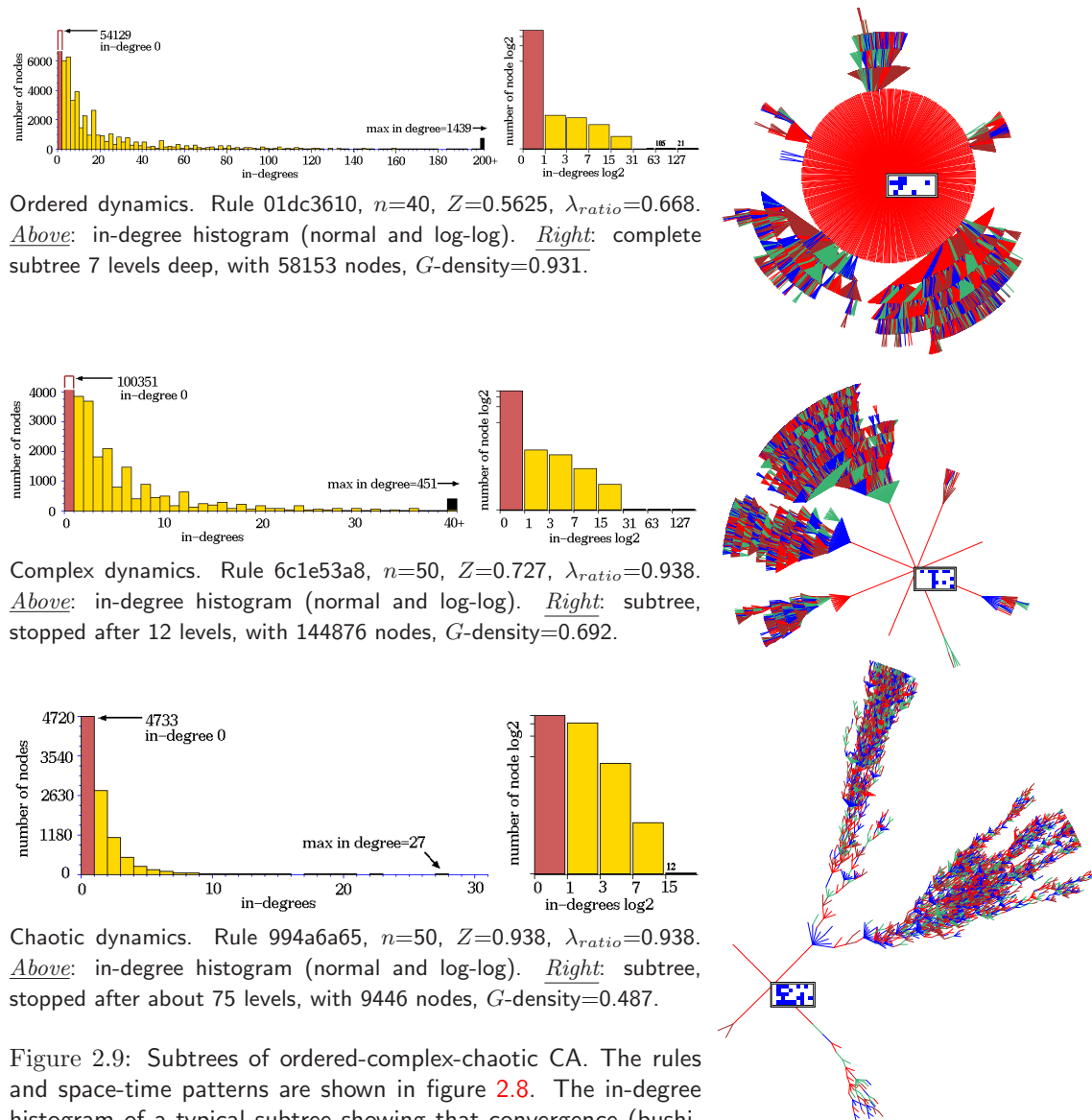


Figure 2.9: Subtrees of ordered-complex-chaotic CA. The rules and space-time patterns are shown in figure 2.8. The in-degree histogram of a typical subtree showing that convergence (bushiness of subtrees) is: ordered-high, complex-medium, chaotic-low.

### 2.19.1 1d CA wiring reverse algorithm

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 — a “rulemix”. This is the most efficient thus fastest algorithm, described in [31, 38]. Furthermore, compression of 1d CA attractor basins by rotation symmetry (section 26.2) speeds up the process.

The inset CAW (cellular automata wiring) will appear in the data window in section 27.2.

### 2.19.2 Nonlocal wiring algorithm

Any other network architecture, with nonlocal wiring, will be handled by a slower *general* reverse algorithm described in [32, 38]. A histogram revealing the inner workings of this algorithm can be displayed. Local 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<sup>1</sup>.

The inset NLW (non-local wiring) will appear in the data window in section 27.2.

### 2.19.3 Exhaustive reverse algorithm

A third, brute force, reverse algorithm first creates a list of “exhaustive pairs” of each state in state-space and its successor (section 29.7) from forward dynamics — this can be saved. The pre-images of states are generated by reference to this list. The exhaustive algorithm is restricted to small systems because the size of the mapping increases exponentially as  $v^n$ , 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 inset EXH (exhaustive algorithm) will appear in the data window in section 27.2.

## 2.20 Random map

The random mapping routine (section 29.8) also creates a list of “exhaustive pairs” as in section 2.19.3 above, but this is done by 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 of this “random map” (with out degree one) are reconstructed by reference to this list using the exhaustive testing algorithm (figure 29.8). The space of random maps for a given system size corresponds to the space of all possible basin of attraction fields and is the super-set of all other deterministic discrete dynamical systems.

The inset MAP (random map, using the exhaustive algorithm) will appear in the data window in section 27.2.

## 2.21 Asynchronous and Sequential updating

By default, network updating is synchronous, in parallel. DDLab also allows asynchronous updating. For space-time patterns (section 31.4) the updating can be sequential, partial order, or noisy. Attractor basins<sup>2</sup> (section 29.9) can also be based on a sequential order. Sequential orders can be forwards, backwards, a random order, or any specific order can be set, out of the  $n!$  possible orders for a network of size  $n$ . The order can be saved/loaded.

<sup>1</sup>Compression does not apply for a local 2d CA with a hexagonal lattice, or for 3d CA, as the algorithms to take account of these symmetries have not been resolved.

<sup>2</sup>Sequential orders do not apply for a range of sizes (section 8.1).

### 2.21.1 Neutral order components

An algorithm in DDLab computes the neutral order components (section 29.10). 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 (figure 29.9.5), or the entire set of components subtrees making up sequence space (the neutral field, figure 29.12) which are drawn in an analogous way to attractor basins.

---

## 2.22 Sculpting attractor basins

Learning and forgetting algorithms (chapter 34) 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 subtrees 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 [32].

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 18.7.4). The inverse problem is — to find a minimal network that will satisfy a full or partial mapping (i.e. fragments of attractor basins such as trajectories).

---

## Chapter 3

# Accessing and running DDLab

This chapter gives instructions for downloading, unpacking and running DDLab, and the various files available. Information is also given about the DDLab web site, manual, GPL license, copyright, and registration. The Linux, Cygwin, Mac, Unix, and Irix versions will be referred to as Linux-like, as opposed to the DOS version.

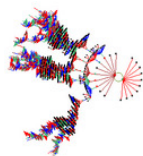
---

### 3.1 The DDLab web site

For the latest compiled versions, source code, makefiles and readme files, and manual, check the DDLab web site located at one of the following,



*Discrete Dynamics Lab*



<http://www.ddlab.org>

<http://www.sussex.ac.uk/~andywu/ddlab.html>

<http://uncomp.uwe.ac.uk/wuensche/ddlab.html>

At the time of writing the latest release was ddlabz04 in April 2016.

---

### 3.2 DDLab at SourceForge

DDLab compiled versions, source code and manual are also available at SourceForge.



<http://sourceforge.net/projects/ddlab/files/>

---

### 3.3 Unzipping and running — Linux-like versions

The relevant readme files, for Linux, Mac, Cygwin and DOS, give the most up-to-date instructions. In general, place the `.tar.gz` file in its own directory, called say, `ddlab`. To unzip and unpack follow the example below (substitute the relevant filename),

```
gunzip ddlabz04_linux64.tar.gz    ... to unzip the .gz file
tar -xvf ddlabz04_linux64.tar    ... to unpack the .tar file
```

This will give the GNU license and the executable file<sup>1</sup> `ddlabz04` — we will use this filename although the executable may be named `ddlabz04_macTiger64`, `ddlabz04_cygwin32.exe`, etc.

To run the program enter

```
./ddlabz04 &    (if dot is in your path ./ is not required)
```

The “&” retains control of the terminal window, where messages and data are sometimes shown. The default background is black — change to white with the program parameter `-w`,

```
./ddlabz04 -w &
```

If working with DDLab file types for rules, seeds, networks etc. (chapter 35), its preferable to run DDLab from a sub-directory (say `ddlab/ddfiles`) containing the `ddextra.tar.gz` files (section 3.6) — in that case enter,

```
../ddlabz04 &    (two dots before the slash)
```

This ensures that files created within DDLab stay within the `ddfiles` subdirectory, and do not clutter up the `ddlab` directory.

### 3.3.1 Unix library files

DDLab for Linux-like versions is compiled with “static” set, so that missing library problems should not occur. However, the libraries `libx11` and `libsunmath` need to be in your system — they usually are. If missing, they can be downloaded from the file `unix_libs.tar.gz`, which will unzip and unpack to give the following files, which should be installed in the same directory as DDLab.

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

---

## 3.4 Unzipping and running - DOS

The latest DOS version of DDLab, `ddlabz04_dos32.tar.gz` can be unzipped with Winzip in earlier versions of Windows. For Windows Vista, open source “7zip” works to uncompress, and there are other tool available. The DOS version may have some drawbacks; for better results install Cygwin/X, (a Linux environment inside Windows) and use the Cygwin version of DDLab, or run the Linux version in VMware Player.

The DOS version will unzip to give the GNU license, two font files, and the following,

```
ddlabz04_dos32.exe ... the program (for example).
dos4gw.exe ... the DOS extender, access to extended memory.
```

---

<sup>1</sup>If the executable permission is missing for some reason, it can be restored with the command `chmod +x ddlabz04`.

Keep all these 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, section 5.2).

In Vista, Windows 7 and later, DOS from the command line is no longer supported, but the DOS version of run, 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 800×600 resolution.
-h ... for 1024×768 resolution.
```

For example, for a white background and 1024x768 enter `ddlabbz04_dos32 -w -h`.

## 3.5 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 to get the flavour of DDLab before reading the detailed manual.

## 3.6 Extra data files

The files in `dd_extra.tar.gz`, common to all platforms, contain data used by DDLab, though DDLab will run without the files. The files should be either in the same directory as the DDLab executable, or in a sub-directory and DDLab run from that sub-directory by prefixing `../` before the executable (section 3.3).

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

### 3.6.1 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 (section 32.6.1). Enter `g` for a random rule, or the next rule in the sequence, depending on the setup in section 31.2.9.

*1d complex rule collections, based on 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` ...  $v \geq 3$  complex rules.

*2d complex rule collections, hexagonal lattice, based on kcode*

`g_v3k6.r_v`, `g_v3k7.r_v` ...  $v=3$  complex rules.

### 3.6.2 Selected 2d complex rules, $v=3$

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



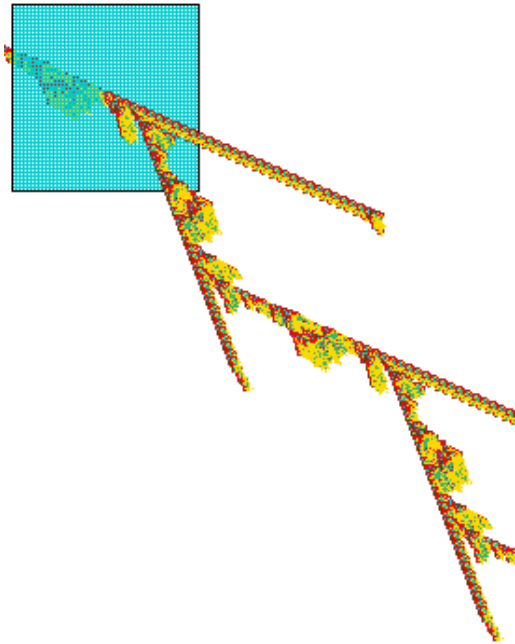


Figure 3.1: A 2-way glider-gun made by rule `v3k6n6.vco` and seed `v3k4gun.eed` shooting gliders in opposite directions, shown as a 2d diagonally scrolling space-time pattern ( $100 \times 100$  hexagonal lattice) in an isometric projection — the present moment is at the bottom right (see also 4.13). Once the glider-gun has shot a glider in one direction, it turns itself inside-out and shoots a glider in the opposite direction. The period between firing successive alternate gliders is 67 time-steps.

`v3k4x1.vco` ... s4-way glider gun (seed `v3k4gun.eed` below).  
`v3k5x1.vco` ... gliders bounce off static structures.  
`v3k6x1.vco` ... the Beehive rule [44], hexagonal lattice.  
`v3k6x2.vco` ... spirals overcome gliders, hexagonal lattice.  
`v3k6B1.vco` ... burning-paper or predator-prey, hexagonal lattice.  
`v3k6n6.vco` ... 2-way glider-gun (figure 3.6.2), hexagonal lattice.  
`v3k7w1.vco` ... the spiral rule [45], hexagonal lattice

### 3.6.3 Selected seeds

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

`pento.eed`, `Lgun_v2.eed`, `Lguns_v3.eed`, `Lguns_v8.eed` ... game-of-Life seeds.  
`v3k4gun.eed` ... seed for 4-way glider-gun (2d or 3d) — `v3k4x1.vco`  
`v3k6n64.eed` ... initial state for 2-way glider-gun — `v3k6n6.vco`

*seeds for the 2d and 3d Beehive-rule — `v3k6x1.vco`*

`Bcgun.eed`, `Bpuff.eed` ... seeds for the 2d 6-way glider-gun, and the amazing puffer-train.

`B3d_ggx.eed` ... seed for the 3d 4-way glider-gun.

*seeds for the 2d and 3d Spiral-rule — `v3k7w1.vco`*

`ssg1.eed`, `sgg2.eed` ... seeds for two types of 2d 6-way spiral glider-guns.

`sgun3d.eed`, `sgun3d1.eed` ... seeds for the 3d 4-way glider-guns.

### 3.6.4 Sorted rule samples

Samples of CA rule-space, sorted by input-entropy and its variability, classify rules between order, complexity and chaos [38]. The samples can be loaded and displayed as scatter plots (section 33.6), and rules can be selected on-the-fly (section 32.6.3) when running space-time patterns. The topic is described in “Classifying rule space” chapter 33. The following sample files are available,

*1d complex rule samples, based on rcode*

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

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

*1d complex rule samples, based on tcode*

v8k5tm.sta ... max-minmax-entropy

*2d complex rule samples, based on kcode*

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

v3k7bs.sta, v8k3B50.sta, v2k5iso.sta, v4k72d52.sta ... max-minmax-entropy.

### 3.6.5 Byl’s self reproducing loop

J.Byl’s self reproducing loop [9], is a  $v6k5$  2d CA, a simplification of Langton’s loop.

v6k5\_by1.rul ... Byl’s self reproducing loop rule (rcode).

v6k5\_by1.eed ... a seed to initiate Byl’s loop.

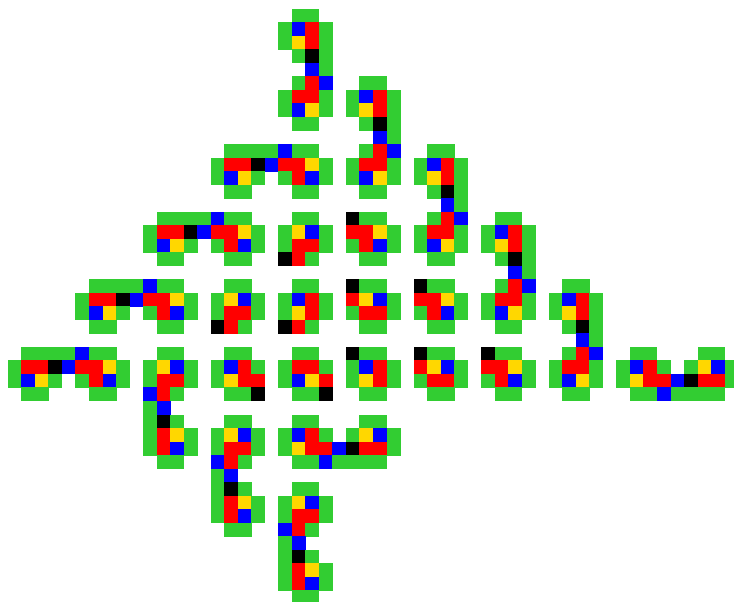


Figure 3.2: Byl’s self reproducing loop — 148 time-steps from this seed:



---

## 3.7 Copyright, License and Registration

### *Copyright*

DDLab is copyright (c) 1993-2016, Andrew Wuensche.

### *License*

DDLab is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License (GPL) as published by the Free Software Foundation, either version 3 of the License, or any later version (<http://www.gnu.org/licenses/gpl.html>), which includes a Disclaimer as follows: This program is distributed in the hope that it will be useful, but without any warranty; without even the implied warranty of merchantability or fitness for a particular purpose.

### *Registration*

By registering and paying a modest fee, the annoying UNREGISTERED banner (section 5.4) can be easily removed — <http://www.ddlab.org/ddinc.html> provides details.

---

## 3.8 Source code, and compiling

The DDLab source code is written in `c`, and consists of about 63000 lines excluding blanks and comments, contained in 21 `*.c` files and two `*.h` files — a description can be found on the DDLab website, including Makefiles and notes on how to comple. The same code compiles for all platforms.

---

## 3.9 Previous versions of DDLab

Previous versions of DDLab released at intervals since 1995 are still available, as are previous versions of the DDLab Manual — refer to <http://www.ddlab.org>.

---

# Chapter 4

## Quick Start Examples

This chapter briefly describes the DDLab graphical user interface (more details in chapter 6), and gives a number of examples of DDLab functions and routines. Try these examples first, to get the flavor of DDLab, before tackling the detailed program reference — chapter 5 onwards.

---

### 4.1 The DDLab screen

In Linux-like systems, or Windows supporting DOS, DDLab starts in 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 within the screen as “windows” or “prompts” — their location is usually indicated, for example “top-right”, “top-left”, “bottom-left”, etc.

If the executable filename is `ddlabbz04`, to run DDLab, for DOS enter `ddlabbz04` at the DOS or command prompt. For Linux-like systems enter `./ddlabbz04 &` — always add a final `&` to retain control of the terminal 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. `./ddlabbz04 -w &` for Linux, `ddlabbz04 -w` for DOS. Descriptions of colors assume a white background.

If an **UNREGISTERED** banner is displayed, enter **return** to continue<sup>1</sup>. 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 pointer 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.

---

<sup>1</sup>Another program parameter turns off the **UNREGISTERED** banner for registered 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 the 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 **Ctrl-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** followed by **return** to skip forward.

### 4.1.5 The graphics setup

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 in Linux-like systems — resize by dragging the corners (or edges) in the usual way. For DOS (section 5.2) the initial size is 640×480, which can be reset to higher resolution — SVGA. To change the graphics setup after DDLab has started, at the first prompt select **g** for graphics (section 6.3). 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.

## 4.2 Basin of attraction fields

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

1. From the first prompt keep accepting defaults with **return** or left mouse button (about 13 presses), until the top-center **basin 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.

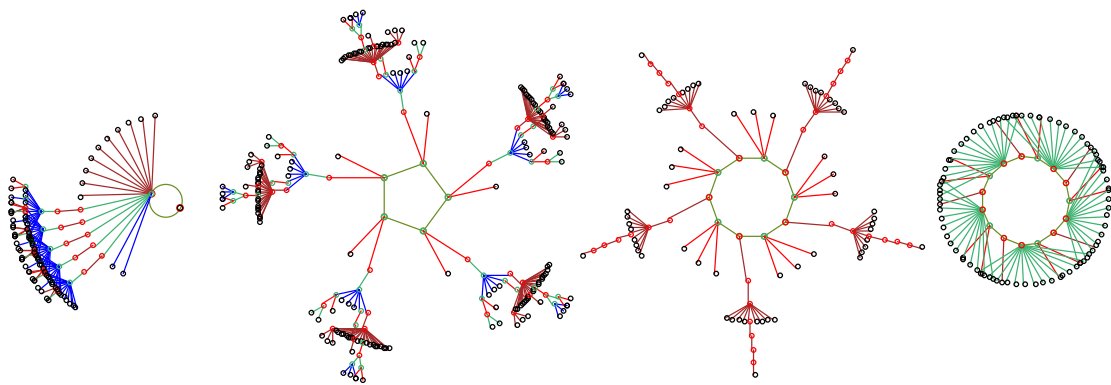


Figure 4.1: A basin of attraction field of a binary 1d Cellular Automaton,  $v2k3$ ,  $n=10$ , decimal rule 9, with copies of equivalent basins suppressed.

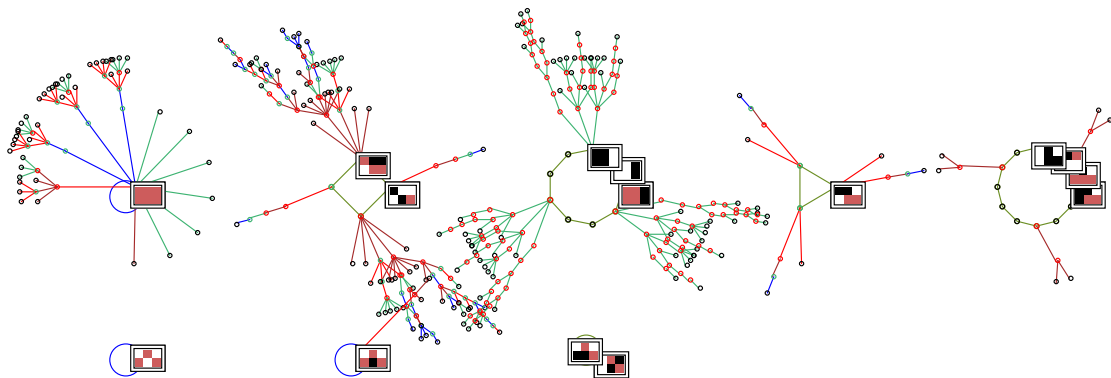


Figure 4.2: The basin of attraction field of multi-value  $v3k3$   $n=6$  1d CA. The lookup table is 120201201020211201022121111 (1886122584a655 in hex). Just the 8 nonequivalent basins are shown from a total of 23, and attractor non-equivalent states are shown as 2d patterns. State-space= $v^n=3^6=729$ . Note that the overlap can be fixed with layout options (chapter 25 or section 20.7).

4. A prompt window appears top-left. Enter **return** for a new basin of attraction field, a one-bit “mutant” of the previous rule, with corresponding data. This process can continue indefinitely (it can also be set on automatic).
5. Enter **q** to interrupt and backtrack up the prompt sequence.

#### 4.2.1 Changing basin parameters

Try the previous routine again, changing  $v$ ,  $n$  or  $k$ , which require backtracking to the main series of prompts. For example, to create figure 4.2 procede as follows,,

1. To revise  $v$  backtrack to 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 k:** ... 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**).

The scale, position, node display etc. can be fine-tuned with the special basin parameters options, which can be accessed one by one, or by jumping directly to a category (chapter 31).

1. At the top-right **revise from:** options, enter **p** for the “display” category.
2. Enter **return** until the prompt **highlight attractor ... :** enter **a** for “all”, then **d** twice to accept further defaults and start drawing basins.
3. Enter **return** for a mutant basin, or **q** to backtrack.

Note that increasing  $v$  will reduce the maximum allowable  $k$  and  $n$  (sections 7.2, 7.3), and as these values increase basins will take longer to generate.

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

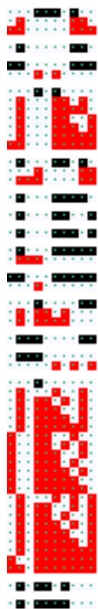


Figure 4.3: Backwards space-time patterns relating to the basin of attraction field of the  $v=2$  CA in figure 4.1. Space across, time top down. The red and white bit patterns are the predecessors of black and white bit patterns.

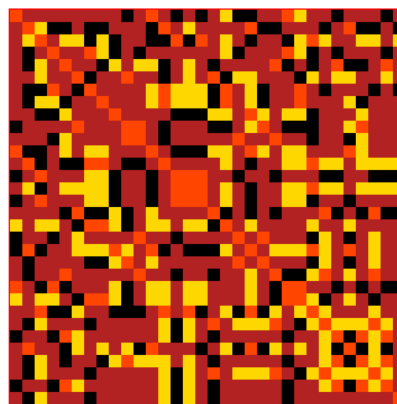


Figure 4.4: The state-space matrix represents state-space, plotting the left half of each state bitstring against the right half. Colors represent different basins of attraction in figure 4.1.

While the attractor basins are generating, various display settings, indicated in the bottom title bar, can be changed on-the-fly. However, basins may generate too fast to intervene on-the-fly. In this case, at the pause when a basin is complete, enter **s** for **speed** in a top-left window, and follow self-explanatory prompts to slow down. Alternatively, backtrack to slightly increase  $n$ ,  $v$  or  $k$ .

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 **#**.

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 **>** to restore maximum speed.
4. Enter **q** to interrupt and backtrack up the prompt sequence.

#### 4.4 Basin of attraction fields for a range of network sizes

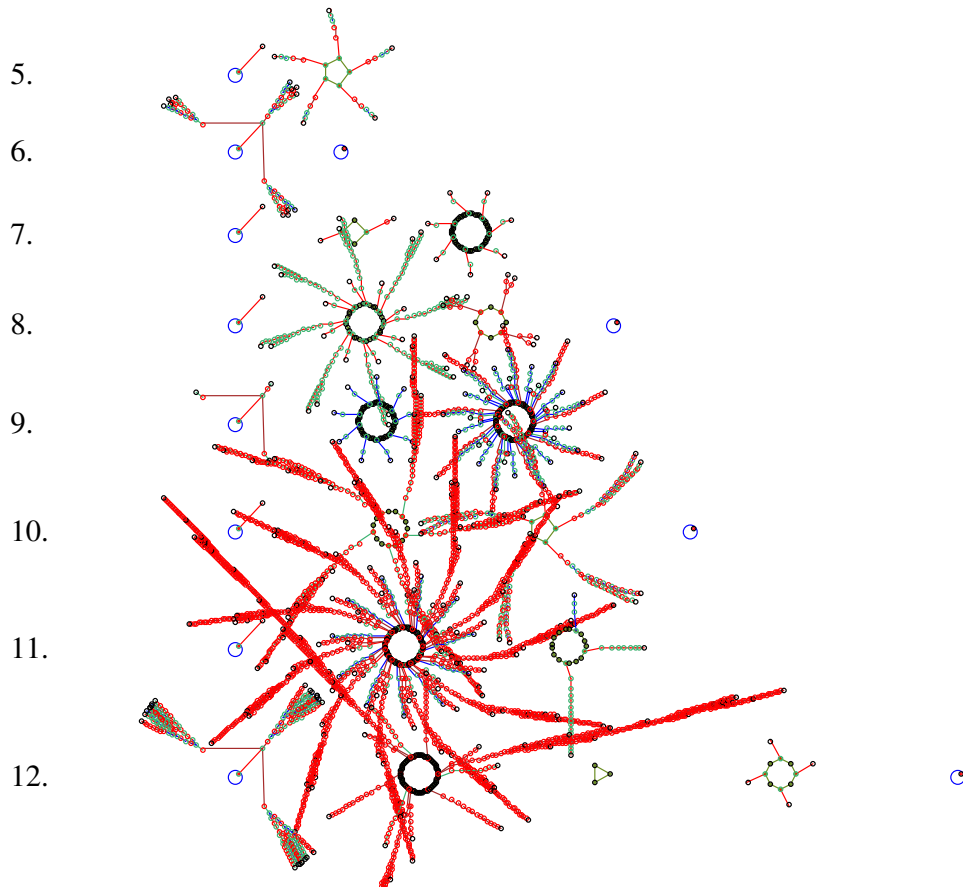


Figure 4.5: Basin of attraction fields for a range of network size  $n=5-12$ .  $v2k3$ , rule(dec)=30

To produce output similar to pages in the “Atlas of Basin of Attraction Fields”, Appendix 2 of the book “The global dynamics of Cellular Automata” [31], 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 basin parameters banner appears, then **a** to restore all defaults, then **d** to skip further special options, then enter **return** to start the *range* of CA basin of attraction fields (with the same rule), for increasing sizes from 5 to 12.



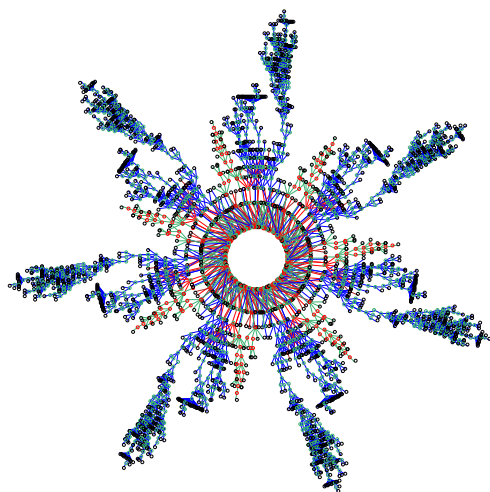
4. When complete, enter **return** for the next “mutant” CA rule.
5. Toggle the display of the “backwards” space-time patterns with **s** and the state-space matrix with **m** as described in 4.3.
6. To slow down the generation of basins (probably required to intervene on-the-fly) enter **s** for **speed** at a pause in the top-left window, and follow self-explanatory prompts.
7. 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 **basin parameters** and enter **l** for layout, then adjust the size and spacing of basins with various self-explanatory prompts (chapter 25).

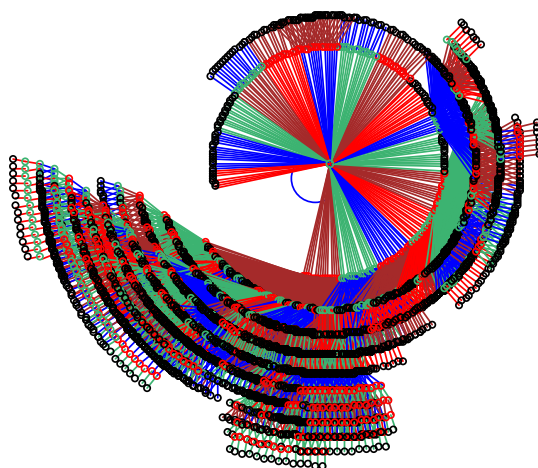
## 4.5 A single basin of attraction

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 k: ...** prompt, enter 4.
3. Enter **return** until the top-center **basin parameters** banner, then **a** to restore all defaults, then **d** to skip further special options, then **return** in response to further prompts, to generate a single basin for a CA, size 14.
4. Enter **return** for the next mutant.
5. Toggle the display of the “backwards” space-time patterns with **s** and the state-space matrix with **m** as described in 4.3.
6. Enter **q** to interrupt and backtrack up the prompt sequence.



states 4333, period 140, rcode(hex) 76b5



states 15541, point attractor, rcode(hex) ac88

Figure 4.6: Examples of single basins of attraction:  $v2k4$ ,  $n=14$ , decimal seed=3187.

## 4.6 A subtree

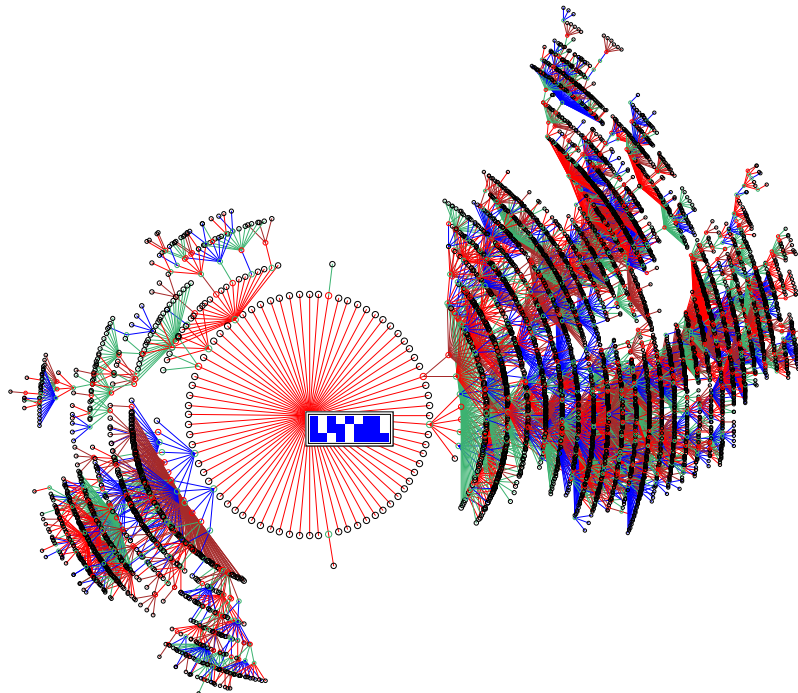


Figure 4.7: A subtree with 11324 states generated from the bit pattern at the center. 1d CA,  $v2k5$ ,  $n=27$  shown as  $9 \times 3$ , hex rcode 5afbb1ae, hex seed 055addaf.

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 k:** ... select 5.
4. Enter **return** in response to further prompts until the **Select SEED ...** : prompt. Enter **r** for a “random seed” then **a** to set *all* cells at random.
5. Enter **return** until the top-center basin parameters banner, 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 subtree.

8. Enter **return** in response to further prompts to generate the subtree. For a deeper subtree, enter a greater number of forward time-steps at the previous prompt. This might

reach an attractor state, in which case the whole basin will be generated with the message **subtree=basin** in the top-right data window.

9. Enter **return** for the next mutant.
10. Enter **q** to interrupt and backtrack up the prompt sequence.

To highlight the “root” state as a bit pattern, backtrack to basin parameters, enter **p** for display, 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 26.3.

## 4.7 Space-time patterns — SEED-mode or TFO-mode

Space-time patterns can be run in either SEED-mode (based on rcode) which also allows single basins, or alternatively TFO-mode, where basin functions (and prompts) are disabled, so space-time patterns become the only possibility, and rules are restricted to totalistic rules expressed as tcode or kcode, with shorter rule-tables than rcode, allowing larger  $[v, k]$ . This is decided at the very first prompt (section 6.2.1), where initially FIELD-mode is active,

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

Enter **s** for SEED-mode — the main sequence prompts continue, or enter **t** for TFO-mode — the first prompt changes to,

**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 accept TFO-mode and continue, or **b** to revert to FIELD-mode.

SEED-mode allows either single basins (the default) or space-time patterns. The choice is made after the main prompt sequence, at the basin parameters banner, and the prompt ... **space-time pattern only-s**.

Entering **s** changes the banner to space-time parameters and presents the space-time pattern options (section 31.1),

**accept all space-time defaults-d**  
**revise from: start/misc-ret updating-u**  
**entropy-e damage-m attractors-a skeletons-s: (skeletons for v=2 only)**

In TFO-mode, these prompts are presented directly, after the main prompt sequence.

## 4.8 1d Space-time patterns

Backtrack with **q** (or right mouse button) to the very first prompt.

1. At the very first prompt, enter **s** for SEED-mode.
2. At the prompt **Network size ... :** select 150.

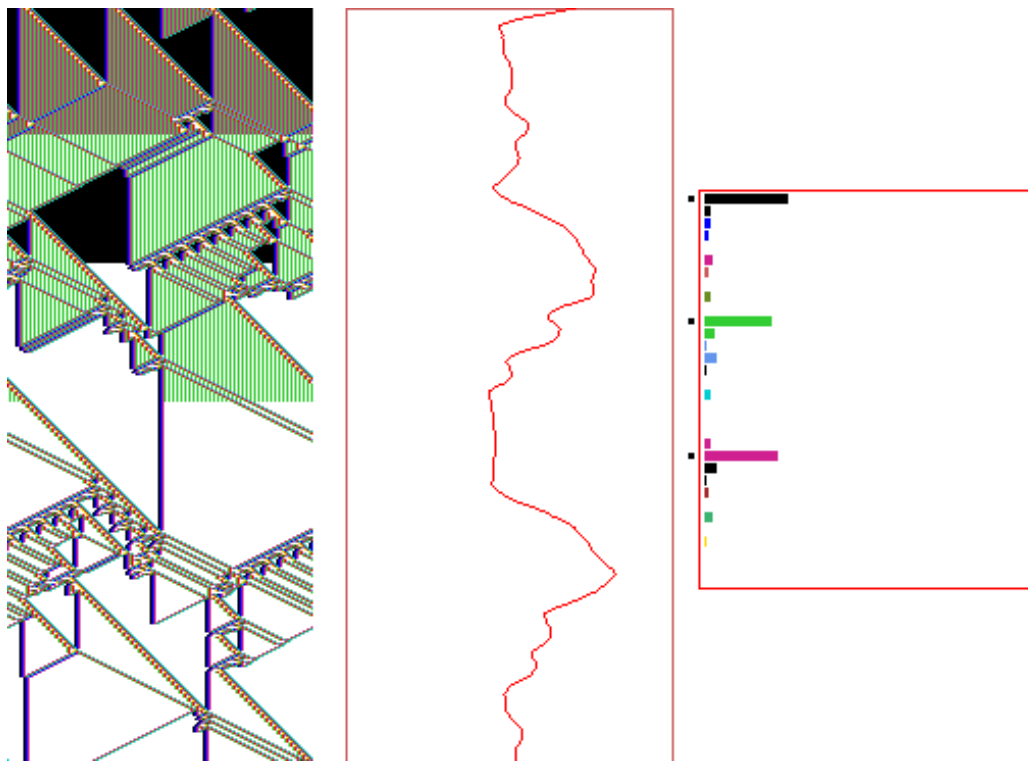


Figure 4.8: A space-time pattern of a 1d  $v2k5$  complex CA, rcode(hex) e9f6a815,  $n=150$ . About 360 time-steps, including some analysis shown by default. *Left*: the space-time pattern colored according to neighborhood, and progressively “filtered” at three times with key **f**, suppressing the background domain to show up “gliders” more clearly. *Center*: the input-entropy/time plot. *Right*: the lookup frequency histogram for the last time step shown.

3. At the prompt **Neighborhood size k:** ... select 5.
4. Enter **return** until the top-center **basin parameters** banner appears, then enter **s** for **space-time pattern only** — the top-center banner changes to **space-time parameters**.
5. At the next prompt, **accept all space-time pattern defaults-d**, enter **d** to skip special options. The 1d 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 hit (section 32.1). Some of these options are repeated on the right of the bottom title bar. Try the following (or any other) 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 key **1**, key **2** will flip them back in reverse order.
  - e/c ... enter **e** or **c** to expand or contract the scale of the space-time pattern.
6. Enter **q** to pause, a top-right prompt appears with further options [32.16](#). For example, to select or revise a particular rule enter **v**. A bottom-right prompt appears, enter **return**, then **h** for the new rule in hex, and enter the hex characters e9f6a815 for the rcode in [figure 4.8](#), then enter **return** until the space-time pattern continues from where it left off, but with the new rule.
  7. Enter **q** to backtrack further.

### 4.8.1 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 32.19](#), [figure 32.37](#)), and the ring itself can be scrolled, making a scrolling tube ([figure 4.9](#)).

To display the ring alongside the normal space-time pattern ([figure 4.8](#)) proceed as follows,

1. Set up the normal space-time pattern as in [section 4.8](#).
2. Enter **q** for the interrupt prompts, then **g** to show the network as a graph, then **return**. A circle of cells will appear.
3. Enter **q** to exit the graph and continue. On-the-fly changes work on the circle as well as the normal space-time pattern.

### 4.8.2 Scrolling the ring

To display the ring and make it scroll, proceed as follows,

1. Set up the normal space-time pattern as above in [section 4.8](#).
2. Enter **T** on-the-fly to toggle the display the 1d space-time pattern to 2d.
3. Enter **q** for the interrupt prompts ([section 32.16](#)), then **g** to show the network as a graph, then **return**. A circle of cells will appear. The appearance/position of the ring can be altered ([chapter 20](#)) but the default setting will be fine.
4. Enter **q** to exit the graph and continue. Space-time patterns will play out as a movie on the ring.
5. Enter **#** or **&** to toggle scrolling the ring.
6. Enter **J** to toggle hiding the scrolling 2d space-time pattern which may be also superimposed on the ring.

Initially the ring will move diagonally towards the bottom-right, then scroll diagonally 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 the following key hits,

- 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 (time-steps).

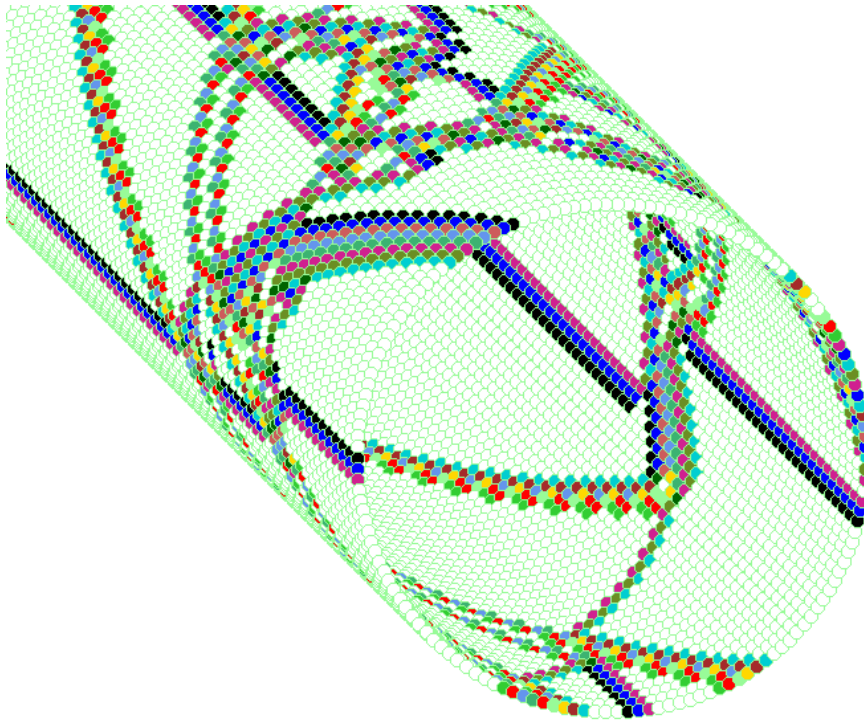


Figure 4.9: A 1d space-time pattern shown as a ring of cells scrolling diagonally upward — a scrolling tube. The present moment is the ring at the bottom-right — the closest ring. The space-time pattern is colored according to neighborhood, and has been filtered. 1d CA,  $v2k5$ , rcode (hex) e9f6a815,  $n=150$ .

- 4 ... for a new random initial state.
- f ... to progressively filter, a to totally unfilter.
- 4 ... for a new random initial state.

### 4.8.3 Multi-value 1d space-time patterns in TFO-mode

Set TFO-mode at the first prompt as described in section 4.7, then procede as follows,

1. At the second prompt, **value-range** ... : select 8, the current maximum in DDLab.
2. At the prompt **Network size** ... : select 150.
3. At the prompt **Neighborhood size k:** ..., select 7.
4. Enter **return** until a top-center space-time parameters banner appears, together with the top-right prompt **accept all space-time pattern defaults-d** — enter **d** to skip special options.

A space-time pattern is generated, scrolling upwards on the left of the screen, as in section 4.8. 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 4.10 Try the following on-the-fly key hits,



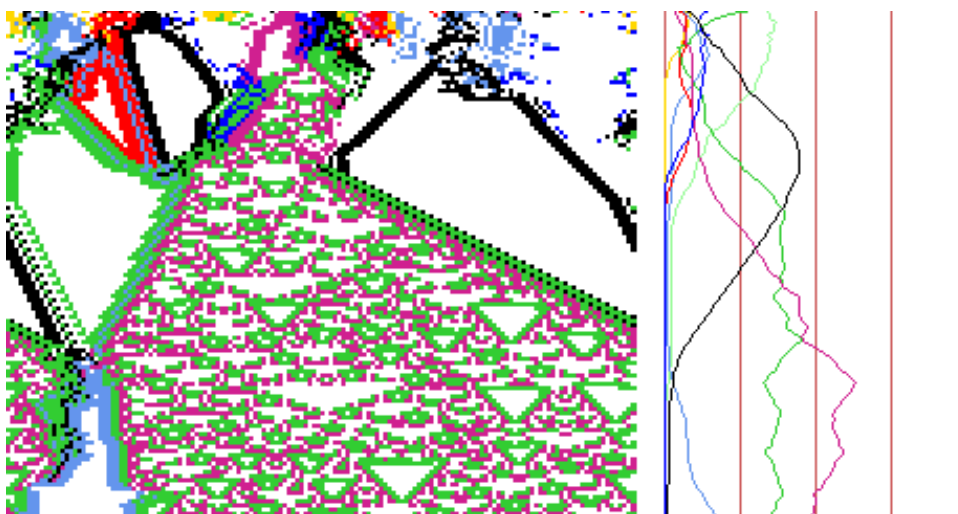


Figure 4.10: A 1d CA with a filtered Altenberg rule, kcode *v8k7*,  $n=150$ , where the probability of a rule-table output depends on the fraction of colors in its neighborhood. On the right the color density is plotted for each of the 8 colors, relative to a moving window of 10 time-steps.

- A** ... for an totalistic “Altenberg” rule, where the output of each neighborhood relates to the fraction of colors in the neighborhood (section 16.9).
- s** ... to toggle between input-entropy and the density plot, the fraction of colors relating to a moving window of 10 time-steps.
- r** ... for a random rule, followed by **A** for another Altenberg rule, and **4** for a random initial state.
- f** ... to filter the space-time pattern repeatedly as required (unfilter with **a**). This highlights domain boundaries (figure 4.10). Filtering applies only if the input-entropy plot is active.
- 5** or **6** ... for a singleton seed, one random cell against **5**() a uniform background of 0s, or **6** against a random but different uniform value (color). Note that the pattern symmetry is conserved however many times the totalistic rule is changed (try with **r**) because totalistic rules are isotropic.

#### 4.8.4 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:

While space-time patterns are running, enter one of the curly brackets on-the-fly, **{** for update probability, **}** for output probability, which toggles between correct updating and 95% of cells updating at each time-step, or 95% correct and 5% updating randomly, where 95% is the initial default setting for both types of noise, which can be combined. Try this with “complex” rules (on-the-fly **g**, section 4.8), and with the “Altenberg” rules (on-the-fly **A**, section 4.8.3).

Changing the default settings is described in section (section 31.4).

## 4.9 2d Space-time patterns

The 2d lattice is defined in the first top-right wiring prompts that appear after **Neighborhood** . . . during the main prompt sequence, starting with,

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

Subsequent top-right prompts reset the network size  $n$  and neighborhood size  $k$  — previous main sequence settings are superseded.

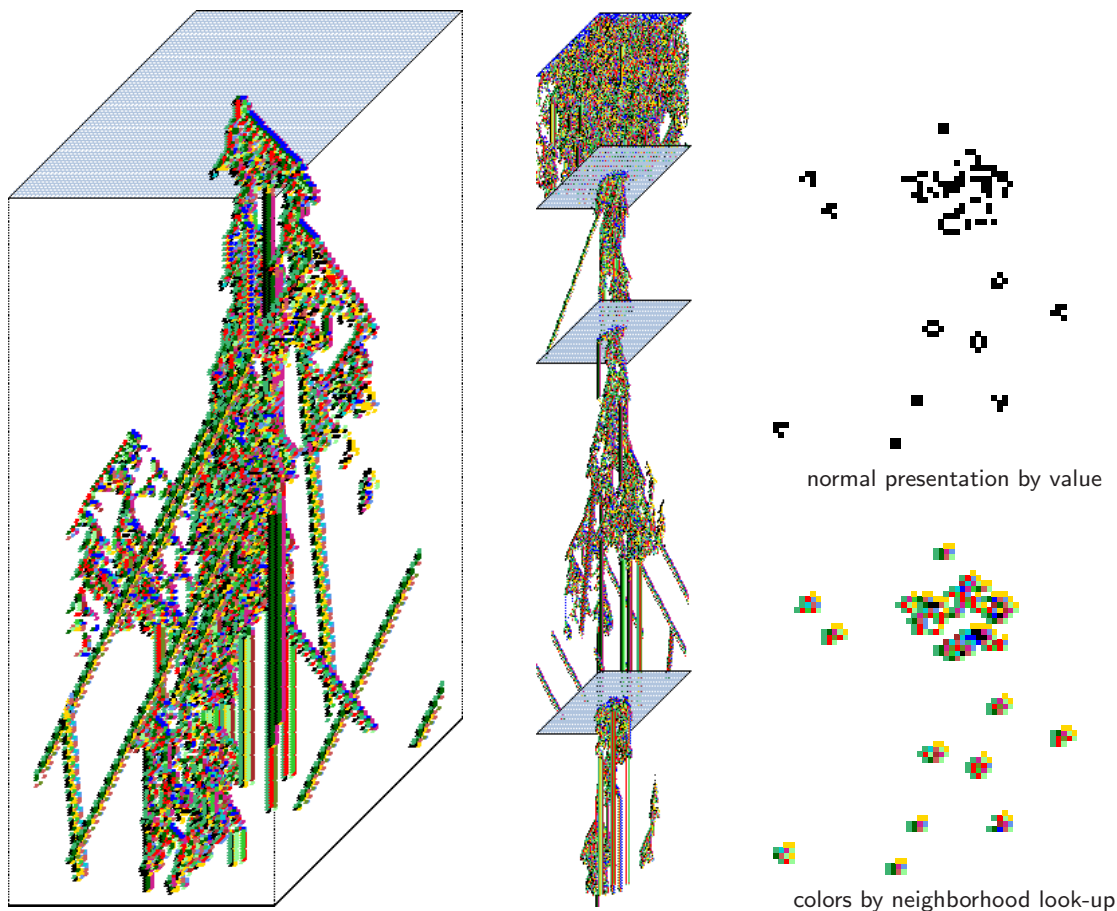


Figure 4.11: Space-time patterns of the 2d game-of-Life, ( $k=9$ ,  $n=66 \times 66$ ) with time-steps stacked below each other in a isometric projection scrolling upwards. *Left:* starting from the “r-pentomino” seed. *Center:* re-scaled to the smallest scale, with new seeds set on-the-fly at intervals. *Upper Right:* The state at time-step 230. *Lower Right:* the same state colored according to the neighborhood look-up instead of the value.



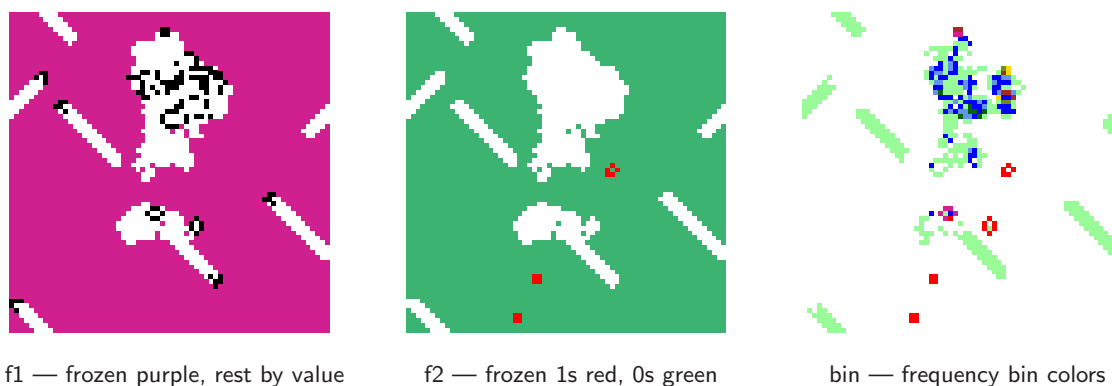



Figure 4.12: Space-time snapshots of the game-of-Life ( $k=9$ ,  $n=66\times 66$ ), time-step 230 as in figure 4.11, but showing three alternative “frozen” presentations set on-the-fly (section 32.11.1). This produces a time-trail behind gliders.

#### 4.9.1 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, enter **s** for SEED-mode.
2. At the second prompt **Value range ...**: enter 2.
3. Enter **return** until the top-right **WIRING:** prompt window appears, and enter 2s for a hexagonal lattice.
4. At the next top-right prompt, **2d, enter width (def-40):** enter 66, which is followed by **depth (def 66):** enter **return**, for a  $66 \times 66$  square lattice.
5. At the next top-right prompt,  
**Neighborhood size k: kmix-m, or enter 1-13 (def 3):** enter 9.
6. Enter **return** until the main sequence prompt for rule selection appears,  
**Select v2k9 rcode (S=512):** ... select *life-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 *empty-e* to “empty” all cells to zero, then *load-l* to load a seed, then enter the filename “pento” at the **LOAD SEED...** prompt. This is the “r-pentomino” pattern  that guarantees gliders. Alternatively, enter *bits2d-b* to draw the seed (section 21.4).
8. Enter **return** until the top-center basin parameters banner appears, then enter **s** for **space-time pattern only**.
9. The top-center banner changes to space-time parameters with the top-right prompt, **accept all space-time pattern defaults-d**, enter **d** to skip 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 on-the-fly key hits (among others) to see what happens.

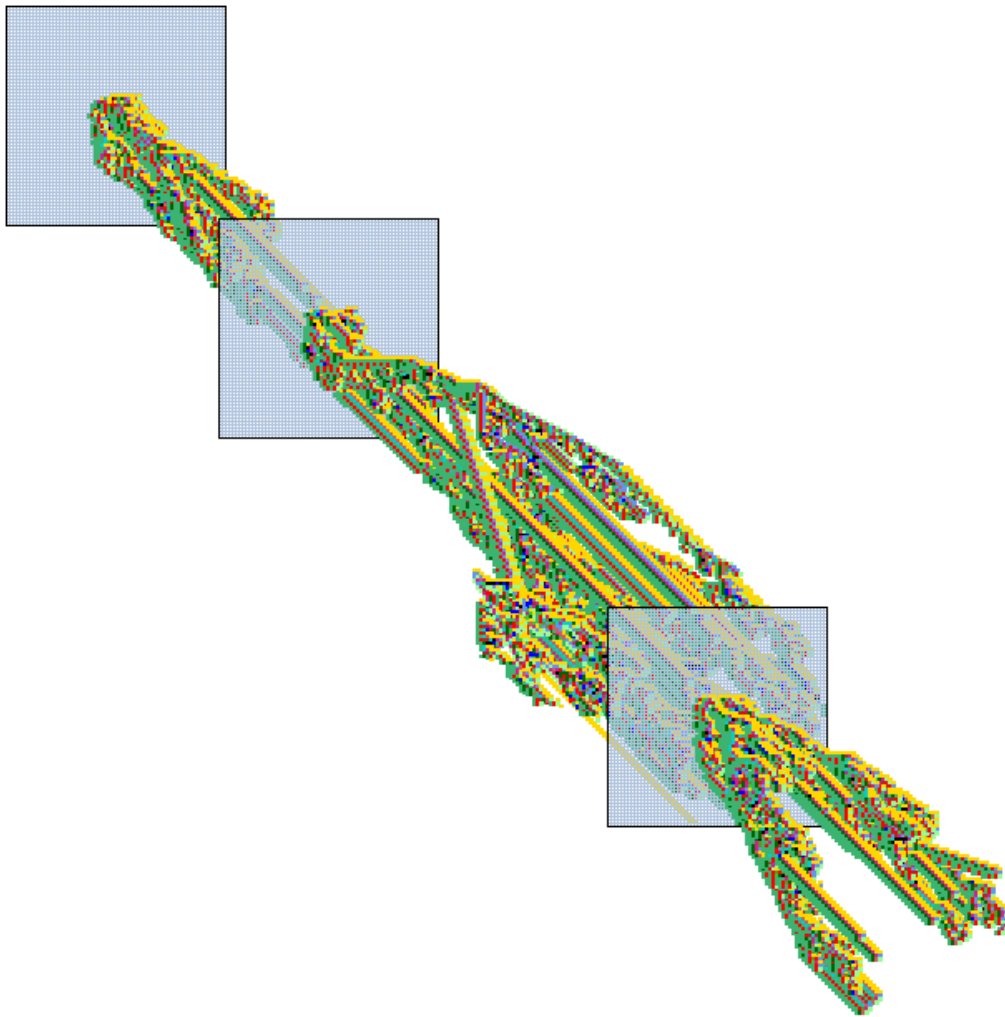
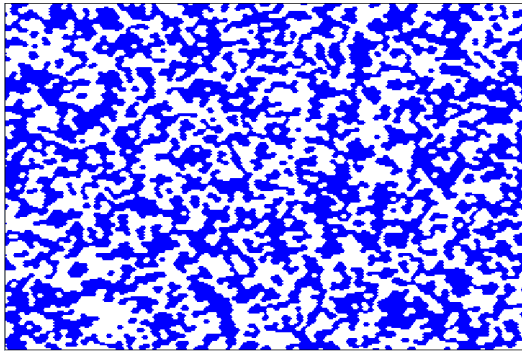


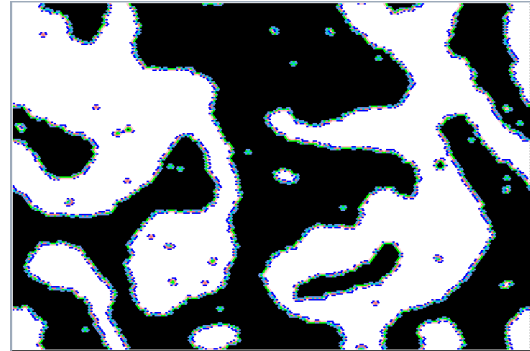
Figure 4.13: Space-time patterns of the game-of-Life ( $k=9$ ,  $n=66 \times 66$ ) consisting of time-steps stacked in front of each other in a isometric projection scrolling diagonally upwards, with new seeds set at intervals, and alternate time-steps skipped.

- 3** ... to toggle cell color — according to the lookup neighborhood or the value.
- h** ... to toggle three ways of displaying “frozen” regions — the stability of the pattern (figure 4.12).
- 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 alternate time-steps.
- k** ... for a new random central block. **4** for a fully random seed.
- e/c** ... to expand/contract the scale of the space-time pattern.

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



(a) *v2k7* majority rule  
tcode=11110000.



(b) *v2k7* modified majority rule  
tcode=11101000.

Figure 4.14: 2d space-time patterns of a *v2k7* CA on a hexagonal grid (240x240), from a random initial state showing aggregating behavior.

#### 4.9.2 2d Space-time patterns — binary totalistic rules

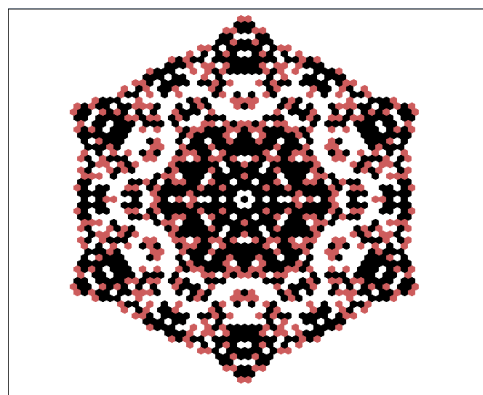
Backtrack with **q** (or right mouse button) and follow steps 1 and 2 as for the game-of-Life (section 4.9.1).

1. Enter **return** until the top-right **WIRING:** prompt window appears, and enter 2x for a hexagonal lattice. Any previous network and neighborhood size settings will be superseded.
2. At the next top-right prompt, **2d, enter width (def-40):** enter 240, which is followed by **depth (def 240):** enter **return**, for a  $240 \times 240$  hexagonal lattice.
3. At the next top-right prompt, **Neighborhood size k: kmix-m, or enter 1-13 (def 9):** enter 7.  
This results in a hexagonal 2d lattice, where each cell has 6 nearest neighbors.
4. Enter **return** until the main sequence prompt **totalistic: tcode-t...:** enter **t**.
5. At the prompt **Select v2k7 rule (S=128) ...:** select select *maj-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 for figure 4.14(a), or first modify the bit pattern to 11101000 (enter 1 or 0, arrow keys to move) for figure 4.14(b).
6. Enter **return** until the prompt **Select SEED (v2 2d ij=66,66)...:** enter *rnd-r* for a random seed, then *all-a* for “all” of the lattice, as opposed to a central block.
7. Enter **return** until the top-center **basin parameters** banner appears, then enter **s** for **space-time pattern only**.
8. The top-center banner changes to **space-time parameters** with the top-right prompt, **accept all space-time pattern defaults-d**, enter **d** to skip 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 previously mentioned key hits, or any others listed.



(a) v3k6 kcode(hex)=0a0282815a0154 giving spiral structures, from a random initial state.



(b) v3k6 kcode(hex)=a0468298295220 (set at random) from a singleton seed.

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

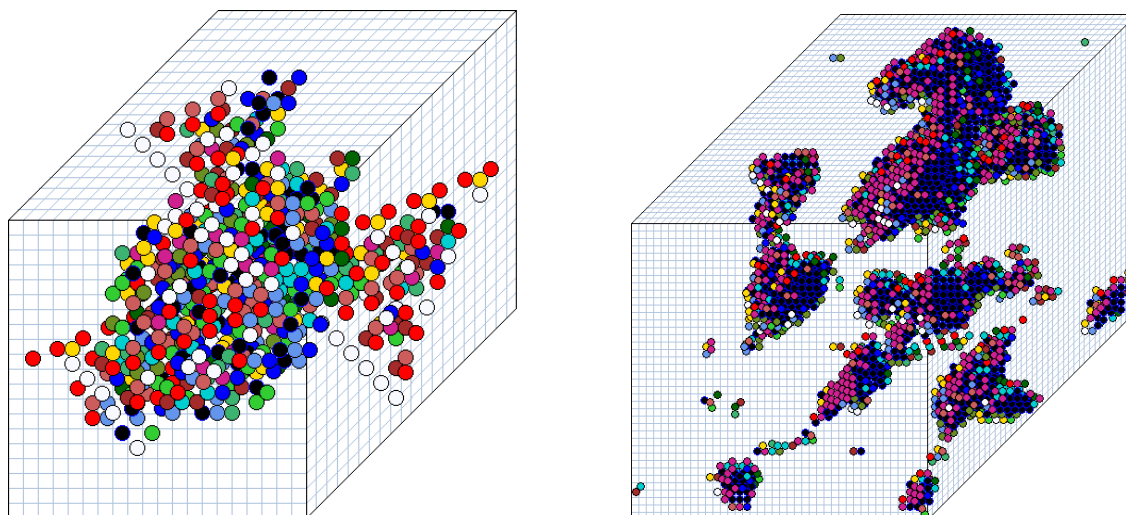
### 4.9.3 Multi-value 2d space-time patterns in TFO-mode

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

1. At the very first prompt, enter **t** for TFO-mode, if not already active.
2. At the second prompt **Value range ...**: enter 3.
3. Enter **return** until a top-right **WIRING** prompt window appears,  
**WIRING: special-s load-l random-r**  
**local: 3d-3, 2d-2(hex+x), 1d-def:** enter 2x for hexagonal 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 240, then at the continuation prompt **depth (def 240):** enter **return**, for a 240 × 240 lattice.
5. 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.
6. Enter **return** until the main sequence prompt  
**totalistic only: outertot-o/+o, tcode-t, (def-kcode):** enter **return** for kcode.
7. 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.
8. Enter **return** until the prompt **Select SEED (v3 2d ij=240,240), ...**: enter **return** for a random central block.
9. Enter **return** until the top-center **space-time parameters** banner appears, with the top-right prompt, **accept all space-time pattern defaults-d**, enter **d** to skip special options.

The 2d space-time pattern is generated in the top-left of the screen. The **on-the-fly key index** appears on the right of the screen. Try the previously mentioned key hits, or any others listed.

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



(a) 3d CA  $20 \times 20 \times 20$ ,  
random  $v2k7$  from a singleton seed

(b) 3d CA  $40 \times 40 \times 40$ ,  
 $v2k7$  tcode=11101000

Figure 4.16: Examples of a 3d  $v2k7$  CA, with a 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 for a clearer picture (instead of by value: 0,1).

(a) a snapshot ( $20 \times 20 \times 20$ ) with a randomly selected rule from a singleton seed.

(b) a snapshot ( $40 \times 40 \times 40$ ) — the maximum size DDLab supports, from an initial state set at random, but with a bias of 45% 1s. The rule is the same as in figure 4.14(b) showing aggregating behavior.

## 4.10 3d Space-time patterns

The 3d lattice is defined in the first top-right wiring prompts that appear after **Neighborhood** ... during the main prompt sequence, starting with,

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

Subsequent top-right prompts reset the network size  $n$  and neighborhood size  $k$  — previous main sequence settings are superseded.

### 4.10.1 3d Space-time patterns — examples

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

1. At the very first prompt, enter **s**, enter **s** for SEED-mode, or **t** for TFO-mode.
2. Enter **return** until the top-right **WIRING:** prompt window appears, and enter 3 for a 3d hexagonal lattice.
3. At the next top-right prompt, enter the width, depth and height.

**3d, enter width (def-9): depth (def 9): height (def 9):**

The maximum cube would be  $40 \times 40 \times 40$ . The lattice has 3-torus boundary conditions.

4. At the next top-right prompt,

Neighborhood size k: kmix-m, or enter 1-13 (def 6): enter 7.

The neighborhood is arranged as a 3d cross.

5. Enter **return** to accept further defaults, including the rule and seed, until the top-center **output parameters** banner appears (for SEED-mode), then enter s for **space-time pattern only**. For TFO-mode this step is omitted.
6. The top-center banner changes to **space-time parameters** with the top-right prompt, **accept all space-time pattern defaults-d**, enter **d** to skip special options.

The 3d space-time pattern is generated in the top-left of the screen. The **on-the-fly key index** appears on the right of the screen. Try the previously mentioned key hits, or any others listed.

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

## 4.11 “Screen-saver” demo

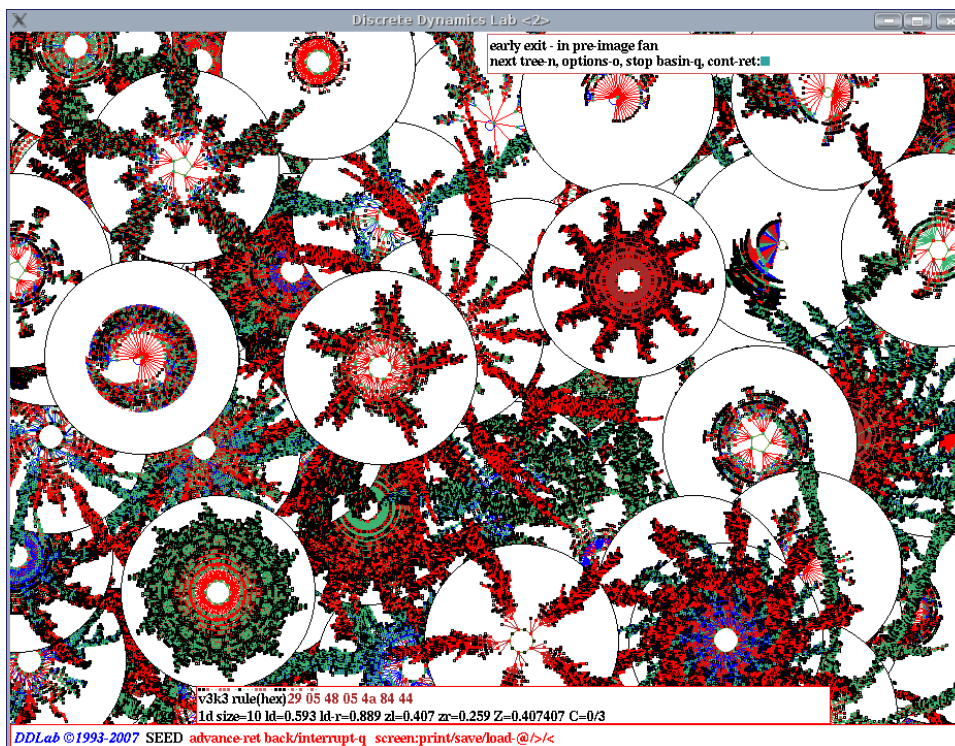


Figure 4.17: Screen-saver demo for Cellular Automata,  $v=3$ ,  $k=3$ ,  $n=10$ . Single basins with rules and initial states chosen at random are generated at random positions in the DDLab screen.



The “screen-saver” demo<sup>2</sup> provides a continuous show of single basins of attraction or subtrees popping up 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, enter **s** for SEED-mode.
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 k: ...** prompt, select 3.
5. Enter **return** until the top-center basin parameters banner appears, then enter *all-a* to restore all defaults, then **return** until the **savescreen demo -s** prompt; enter **s** to accept the option.
6. Then enter **d** to accept all further defaults, then **return** twice — the demo will start. The rule picked at random will be shown in the bottom rule-window.

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** (section 4.3). Enter **q** to interrupt and backtrack up the prompt sequence. Try the savescreen demo with other setting of  $v$ ,  $n$ , and  $k$ .

## 4.12 RBN and DDN

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  $v \geq 3$ ), departs from CA network architecture by allowing each cell in the network to have different nonlocal wiring, a different rule, and a different number of inputs,  $k$ , 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 nonlocal **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—4.11 can be adapted as described

<sup>2</sup>See also section 24.8

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 (section 1.6.1), 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 tackled with caution.

1. For random wiring (chapter 12), at the first top-right wiring prompt,

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

2. select **r** for 1d random wiring — no further setting required.
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 square+s), 1d-def:** enter **2** for 2d.
  - (b) **hand wire-h,**  
**local 2d-2 (hex+x square+s), 1d-1, random-def:**  
 enter **return** for random wiring.
  - (c) **2d: square edges, safe<255 max=2896 incr with caution!**  
**enter i (def 40): j (def 40):** enter **return**, or set the size.
  - (d) **Neighborhood size k: kmix-m . . .** select  $k$  (try 5), or enter **m** for mixed- $k$ , in which case there will be a series of further prompts to set the  $k$ -mix (section 9.3), but just enter **return** for a random  $k$ -mix.
  - (e) **bias random wiring:** . . . these further options (section 12.5) can be skipped with **d**.
  - (f) Note: any new entries here for  $n$  and  $k$  will supersede earlier entries.
4. Whatever wiring is defined, local or random, in any dimension, the next top-right prompt (similar to below) allows the wiring to be examined in detail, and amended (section 17.1),

**2d network (122x122), review/revise, wiring only - rules not set**  
**graph-g, matrix: revise-m view-M prtx-Mp**  
**graphic:1d+timestep-1 circle-c 2d-2:**

5. After **totalistic:** . . . in the main sequence of prompts, a top-right window appears with options for a single rule or a rulemix (chapter 14),

**RULES: single rcode (def), load rcodemix-l, list Post-P, nhood-matrix-a**  
**mix: no limit-n, or set limit up to 200:**

Enter **n** for a rulemix, or **return** for a single rule.

6. If a random  $k$ -mix is required, stepping back with **q** to the main prompt sequence prompt,

**Neighborhood size k: kmix-m, or enter 1-13 (def 3):** enter **m**.

Various further options to set and bias the  $k$ -mix are presented (chapter 9). If in doubt enter **return**. For a random  $k$ -mix for a 2d or 3d network, first select the random wiring in 2d or 3d as above.

7. Once rules have been set in the main prompt sequence, top-right options are presented to examine and amend the resulting network in detail, wiring and rules, similar to this (also applicable to CA),



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

- Enter **return** to skip and continue, **q** to backtrack.
- Enter **1**, **c**, **2** (or **3** for 3d) to review the network as a wiring graphic, using the arrow keys and mouse clicks to examine the wiring and rules of chosen elements (chapter 17).
- Enter **m** or **M** to review the network as a wiring matrix (section 17.2)
- Enter **g** to review the network as a graph, which can be rearranged by dragging nodes with the mouse (chapter 20).
- The options above have their own follow-up options described in the section listed.

Having set up the RBN or DDN, follow the guidelines in previous sections in this chapter to generate attractor basins (for example figures 2.5, 2.6, 8.7), and space-time patterns (for example figures 2.7, 16.9, 32.23).

---

# Chapter 5

## Starting DDLab

This and the following chapters provide the detailed program reference, listing and explaining all the various options and prompts with examples of DDLab’s behavior. Before tackling this part of the manual, its a good idea to try some of the “Quick Start Examples” in chapter 4 to get a feeling and flavour of DDLab. 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 directory containing the DDLab executable files, or in the appropriate sub-directory (section 5.1). The executable files downloaded from the DDLab website will be similar to the following,

```
ddlabz04 ... for Linux-like operating systems
ddlabz04_dos32.exe ... for DOS. dos4gw.exe and *.fon files must be in the same directory.
```

The files in `dd_extra.tar.gz` are not essential for DDLab to run, but are required for some functions. They can be placed 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 (chapter 35).

The latest instructions are provided in “readme” files such as `dd_linux_readme_z04.txt`, `dd_cygwin_readme_z04.txt`, `dd_mac_readme_z04.txt`, and `dd_dos_readme_z04.txt`.

---

### 5.1 Running in Linux-like operating systems

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

```
ddlabz04 & ... if “dot” is in your path.
./ddlabz04 & ... if “dot” is not in your path.
../ddlabz04 & ... from a subdirectory of the DDLab directory (containing the files in
dd_extra.tar.gz).
```

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

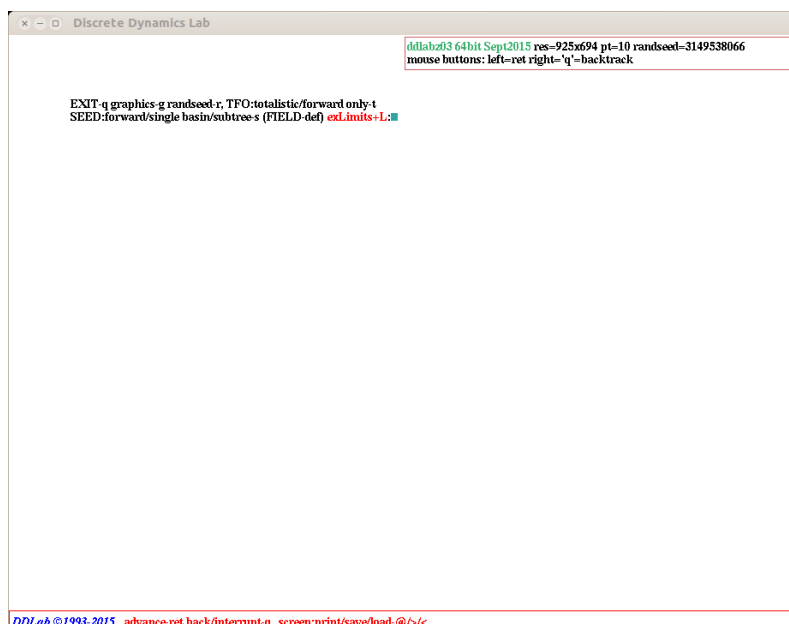


Figure 5.1: Typical graphical user interface when starting DDLab, with the command line argument `-w` for a white screen.

The DDLab screen will appear, similar to figure 5.1 with an initial resolution of  $925 \times 694$ . The message “Using backing store” will appear in the terminal window indicating that the DDLab screen will be restored in the usual way when covered by other windows or moved off the screen<sup>1</sup>.

### 5.1.1 Linux within Windows — Cygwin

Compiled versions of DDLab are provided for Cygwin (CygwinX), a freely available Linux-like environment for Windows. For notes on running DDLab in Cygwin see the Cygwin readme file.

### 5.1.2 Linux within Windows — VMware Player

VMware Player allows Linux Ubuntu (and other Linux distributions) to be setup as a “Virtual Machine” within Windows. Within this, the compiled DDLab Linux versions provided will run in the usual way, or the source code can be recompiled if necessary.

---

## 5.2 DOS within Windows

DDLab is still compiled for old fashioned DOS (MS-DOS) in the Watcom compiler, now open source and freely available as Open Watcom 1.9 (<http://www.openwatcom.org>). DOS operation differs according to the chronology of Windows versions, roughly as follows,

---

<sup>1</sup>However, if DDLab is iconized, resized, or if the “desktop” (or “workspace”) is changed, the immediate current contents in the screen will blank out, but the contents will reappear as DDLab continues.

- In Windows 98 or prior, DDLab can run in pure DOS. There is an option “Restart the computer in MS-DOS mode” where DOS takes over the whole screen. The resolution might need to be changed to correspond to the monitor and graphics driver, otherwise fonts might appear distorted (sections 5.7, 6.3.2).
- In later editions of Windows but prior to Vista and Windows7, DDLab can run from a DOS “command line” window in low resolution, but this window can expand to full screen — **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 window, the resolution must be changed back to “low” to avoid Windows warnings and unforeseen consequences. In higher resolutions than 640×480 the mouse pointer will probably not be visible.
- In Vista, Windows7 and later, DOS no longer exists at the command line. However, DDLab will run (slowly) in DOSBox, an open source MS-DOS emulator, intended for antique PC games. For notes on running DDLab in DOSBox see the DOS readme file.

Enter `ddlabz04` or the name of the DOS executable without the `.exe` extension. Alternatively, from a subdirectory (`ddfiles` — containing the files in `dd_extra.tar.gz`)<sup>2</sup> of the DDLab directory, enter `..\ddlabz04`. This will keep files generated by DDLab in the `ddfiles` subdirectory.

Note that the Linux-like versions perform better, and are better supported, than the DOS version, so Linux within Windows, either Cygwin (section 5.1.1) or VMware (section 5.1.2), are better options than DOS.

---

### 5.3 Command line arguments

Command line arguments for various settings can entered after the executable name — first a space, then a dash (-), then the arguments, without spaces, in any order, for example,

```
./ddlabz04 -wt & ... for Linux-like systems, where & is the final entry.
```

The following command line arguments are available,

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

Additional commands for DOS only,

```
l ... low resolution (VGA) 640×480
m ... medium resolution 800×600
h ... high resolution (SVGA) 1024×768
```

These setting can also be reset once DDLab starts.

---

<sup>2</sup>It may be necessary to copy the `*.fon` files into the `ddfiles` subdirectory.

---

## 5.4 The UNREGISTERED banner

On start-up (and also before exiting) an unregistered version of DDLab will first display the following “UNREGISTERED” banner in the center of the screen. Press **return** to continue. By registering (section 3.7) this annoying banner can be easily disabled.

**UNREGISTERED, see [www.ddlab.org](http://www.ddlab.org), continue-ret:**

---

## 5.5 Title bar

On start-up of a registered version (or if **return** is entered at the “UNREGISTERED” banner) DDLab’s graphical user interface appears as described in figure 5.1. A title bar is displayed across the foot of the screen, including a copyright reminder, the current mode if set (TFO, SEED, or FIELD), and some reminders of key presses which can be activated whenever the prompt cursor is flashing.

*DDLab* ©1993-2016 **advance-ret** **back/interrupt-q** **screen:print/save/load-@/>/<**

*key press reminder* ... *what it means — permanently displayed*

**advance-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.

The title bar displays other on-the-fly options while generating attractor basins (section 30.3), or space-time patterns (section 32.2).

---

## 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 > is entered*)  
 or  
**LOAD SCREEN-filename(no ext) .nat will be added** (*if < is entered*)  
 then  
**change dir-d, now: /home/andy/sussex/ddlabx11:** (*for example*)  
**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 extension (**.nat**) which is added automatically, or accept the default filename **myimage**. There are further details on these filing options, changing directory and listing files, in chapter 35.

When loading a **texttt\*.nat** file there is a further option,

**filename=myimage.nat REVISE-q keepscreen-k cont-ret:** (*for example*)

Enter **return** to overwrite the existing screen with **textttmyimage.nat**, or **k** to add it on top. 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 bitmap PostScript **\*.ps** file<sup>3</sup>. 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, where 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 cross-hair, **+** or **×**. Make sure this cursor is within DDLab and click the left mouse button. The DDLab screen image will be saved as a bitmap PostScript file called **temp\_window\_file.ps** (size about 285k) in the current 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<sup>4</sup>.

The following messages might show up in the terminal window,

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



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.

The last indicates that the size of the DDLab screen was too large for the printer paper size, 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 save/print the DDLab screen, or a selected parts, is with XV<sup>5</sup>, or another screen grabber. XV can grab, save, scale and print the image in one of the many file formats, including GIF and bitmap PostScript. available.

<sup>3</sup>A PostScript file can be viewed with GhostView, in a terminal enter `texttgv filename.ps`.

<sup>4</sup>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`.

<sup>5</sup><http://www.trilon.com/xv/> (enter `xv &` in the terminal)

### 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 DDLab screen appears similar to figure 5.1, with information in a top-right similar to this,

*Linux-like systems*

```
ddlabz04 64bit April2016 res=925x694 pt=10 randseed=3149538066
mouse buttons: left=ret right='q'=backtrack
```

*DOS — screen resolution is 640×480, 800×600 or 1024×768*

```
ddlabz04 32bit April2016 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,

```
ddlabz04 ... the version of DDLab — check the latest at http://www.ddlab.org
32bit ... 32bit CPU, or 64bit CPU. 32bit works on a 64bit CPU but not vice-versa.
April2016 ... 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=24081 ... the random number “seed”, 32bit CPU unsigned short int, 64bit CPU
unsigned long int. The random number seed changes each time DDLab is
run, and 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.
```

In Linux-like systems the default DDLab screen resolution is automatically set to 925×694, but can be resized by dragging with the mouse in the usual way, or set to an exact size (section 6.3.1). In either case the text size will attempt to resize according to the screen width but can be reset manually

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 DOS, the resolution should be reset to higher (or lower) only when DDLab in full screen mode.

---

## 5.8 The mouse pointer in DOS

In DOS and Windows the mouse pointer 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 filename, 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/DDN, 2d and 3d networks, network architecture graphics, analytical measures, data collection, presentation settings, and other special settings. Note that for 2d and 3d CA, and RBN/DDN, the main sequence prompts for the network “wiring” and neighborhood size are superseded in the first pop-up window, labelled “WIRING”.

---

## 5.11 Exit DDLab

To exit DDLab immediately in Linux-like systems, enter **Ctrl-q** at any prompt. Otherwise continue backtracking with **q** (required for DOS) beyond 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.

---

# 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), and a new “exLimits” option (section 6.2.4) for extra limits which allows greater (but risky) neighborhood size  $k$ , and network size  $n$  (sections 7.2 and 7.3). The first prompt also provides pop-up windows to amend the graphics, and the random number seed.

---

### 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 rule-tables (kcode and rcode) are much smaller than full rule-tables (rcode). All attractor functions, which depend on rcode (and their prompts) are disabled in TFO mode. These constraints reduce memory load, allowing larger neighborhoods (section 7.2). 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 (other than TFO-mode) 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 FIELD/SEED-mode setting can also be swapped when attractor basins are complete (section 30.4). Note that totalistic rules, kcode and tcode, can also be selected in FIELD and SEED-modes, but in this case the equivalent rcode will be applied.

## 6.2 The first prompt

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

```
EXIT-q graphics-g randseed-r, TFO:totalistic/forwards onty-t
SEED:forward/single basin/subtree-s (FIELD-def) exLimits+L:
```

At the same time, a top right DDLab information window appears as described in section 5.7.

### 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:) exLimits+L:
```

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 (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 FIELD-mode (the default) to generate a “basin of attraction field”, which 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 for SEED-mode, to generate any of the following, which do require a predefined seed,

- *forward* space-time patterns, starting from a seed.
- *single basin* of attraction, containing a seed.
- *subtree* running backwards from a seed, or from a state downstream from the seed by a predefined number of time-steps.

The seed will be set at later stages in the prompt sequence (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 set (TFO, SEED, or FIELD) will appear in the bottom title bar (section 5.5), for example,

```
DDLab ©1993-2016 FIELD advance-ret back/interrupt-q screen:print/save/load-@/>/<
```

## 6.2.4 The exLimits option

To activate the “exLimits” (extended limits) option and allow greater (but risky) maximum sizes of both the network  $n_{Lim}$  and the neighborhood  $k_{Lim}$ , enter **L**, or **sL** for SEED-mode, at the first prompt (section 6). The resulting maximum sizes depend on your CPU and DDLab version<sup>1</sup>, 32-bit or 64-bit. The value-range  $v$  to be selected in section 7.1 also affects  $k_{Lim}$  (section 7.2), and  $n_{Lim}$  for FIELD-mode (section 7.3). For SEED-mode and TFO-mode  $n_{Lim}$  is independent of  $v$  (section 8.3). The following top-right notice (and prompt) is presented,

*for SEED-mode, TFO-mode, or a 32-bit DDLab and CPU*

**caution! exLimits selected high values of k or n might exceed memory**

*for FIELD-mode and a 64-bit DDLab and CPU*

**caution! exLimits selected high values of k or n might exceed memory**

**enter total RAM+SWAP (def 4.00 GiB):** *(for a 32-bit CPU 0.5 GB)*

With exLimits active, higher limits for the neighborhood size  $k_{Lim}$  will be offered in section 7.1.1.

For FIELD-mode with a 64-bit CPU,  $n_{Lim}$  (section 7.3) can be increased from the basic limit of 31 depending on available memory — random access memory (RAM) plus swap-space (SWAP). The default RAM+SWAP available is assumed to be 4.0 gigabytes. Enter **return** to accept the default, or enter a different measure of gigabytes for your computer<sup>2</sup>. The available memory and the value-range  $v$  will set  $n_{Lim}$ . For example, for 4.0 gigabytes and  $v=2$ ,  $n_{Lim}=35$ . To increase  $n_{Lim}$  by one usually requires doubling available memory.

## 6.3 Graphics setup

Enter **g** in section 6.2 or 6.2.1 to change the window size, background color, font size and line spacing (section 6.4), or flashing cursor speed (section 6.5). A new graphics screen shows the current font and color palette. A top-right window gives prompts as below<sup>3</sup>. When the graphics changes are complete the program will revert to the first prompt in section 6.2.

*Unix/Linux*

**window size=925x694 resize/restore-w/W**

**character width=9 height=15 pointsize=10 revise/restore-s/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:**

<sup>1</sup>32-bit DDLab runs on a 64-bit CPU with 32-bit maximum sizes. 64-bit DDLab will not run on a 32-bit CPU.

<sup>2</sup>Memory information for Linux or Mac can usually be found by entering “cat /proc/meminfo” in a terminal. One gigabyte (GB) =  $1024^3$  bytes, also known as a gibibyte (GiB).

<sup>3</sup>The same graphics setup prompt is available when running attractor basins (section 30.4) or space-time patterns (section 32.17).

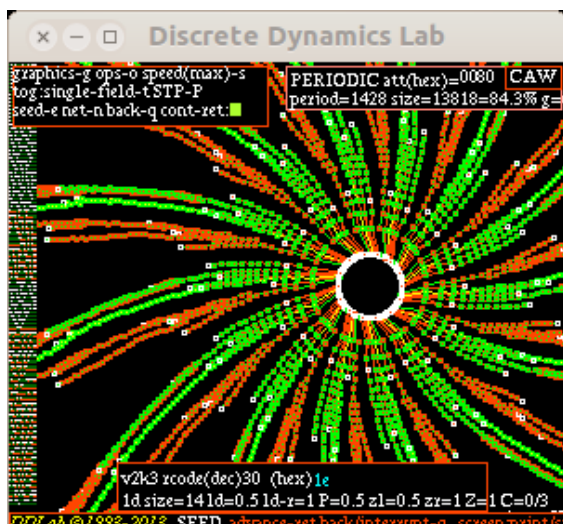


Figure 6.1: The smallest DDLab screen in Linux,  $300 \times 250$  pixels, that can be set from the graphics setup prompts. It is really too small to contain prompts and data. This example has the default black background.

In Linux-like systems, the initial resolution of the DDLab screen is set at  $925 \times 694$ , 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 screen resolution, Linux-like systems

Enter **w** in section 6.3 to resize the DDLab screen to a particular size in pixels, or **W** to restore the start size  $925 \times 694$ . The following top-right prompt is presented,

```
new window size (start=925x694, min 300x250, max=1366x768)
width (default 925): (for example)
height (default 694):
```

The smallest size allowed by this method is  $300 \times 250$  shown in figure 6.1 and the maximum is the monitor size. The window can also be resized by maximising, or dragging a corner with the mouse in the usual way. Note that when the window is resized the font size is automatically changed to an appropriate size, but can be revised (section 6.4).

### 6.3.2 The DDLab screen resolution in DOS

Section 5.2 describes the DDLab screen 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: $640 \times 480$ , **m**=medium: $800 \times 600$  and **h**=high: $1024 \times 768$ . Take care to reset 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. enter `dclabm06 -h` to start in high resolution.

### 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 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 turn,

```
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. Enter **S** to restore all the original defaults.

### 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 default cursor flash speed is set automatically at about 0.5 seconds independent of your CPU speed. However this is not always reliable — to adjust the flash rate enter **f** in section 6.3. The following top-right prompt is presented,

**change flashing cursor speed(1.0): restore-r(1.0) or enter factor:**

Enter a factor to slow down or speed up the current flash rate. For example to make it twice as fast as the current speed enter **.5**. For 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 0-4294967295 (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.

---



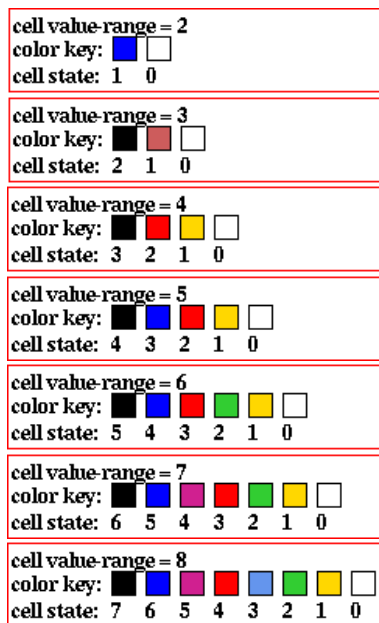
# Chapter 7

## Value-range $v$ , and $n, k$ limits

Figure 7.1: The cell value color key top-left window appears when the value-range is selected, and also at other times, with the colors indexed from  $v-1$  to 0.

*Right:* The color key windows for value-range  $v = 2$  to 8, on a white background.

*Below:* Colors are different on a black background (section 6.3.3) — the  $v=8$  black background color key.



The “value-range”,  $v$ , available to a cell or network element, can be set from 2 to 8, which can be thought of as the number of its potential internal states, colors, or the size of its “alphabet”. Greater  $v$  limits the maximum neighborhood size  $k_{Lim}$  (section 7.2), and also, for basin of attraction fields, the maximum network size,  $n_{Lim}$  (section 7.3) — this chapter describes the basic and extended limits. Note that for SEED-mode and TFO-mode,  $n_{Lim}$  is independent of  $v$  (section 8.3).

---

### 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):** ( $v=2$  on startup — then defaults to the last  $v$  that was 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.

### 7.1.1 extending $k_{Lim}$ with exLimits

If “exLimits” was activated in section 6.2.4, after the value-range prompt in section 7.1 above, the following top-right prompt is presented to set an extended maximum neighborhood size,  $k_{Lim}$ ,

**change -/+ max neighborhood (kLimit): increase with caution!**  
**enter new kLimit (def 13, max 27):** *(for basic  $v=2$ , FIELD mode, and 64bit CPU)*

The maximum  $k_{Lim}$  (and the default) depends on the value-range  $v$ , the current mode — FIELD, SEED or TFO, and your CPU and DDLab version, 32-bit or 64-bit, as set out in table 7.1.

For FIELD-mode and high values of  $k_{Lim}$ , a top-right notice will monitor the progress of setting up the rcode  $k$ -matrix (sections 13.3 and 13.5), for example,

**setting up k matrix: abort-q: 32%**

For higher values of  $k_{Lim}$  this can take some time<sup>1</sup> — enter **q** to abort and revert to the “extending limits” above.

## 7.2 Limits on neighborhood size, $k_{Lim}$

The upper limit of neighborhood size ( $k_{Lim}$ )<sup>2</sup> decreases with increasing value-range  $v$  because rule-tables become too large. The size,  $S$ , of a full rule-table (rcode) is given by  $S=v^k$ , which is larger than a  $k$ -totalistic rule-table (kcode) where  $S=(v+k-1)/(k!(v-1)!)$ , so  $k_{Lim}$  is bigger in TFO-mode (section 13.2.1).

Other factors affecting  $k_{Lim}$  are “exLimits” (extended limits), the option that allows high values of  $k$  and  $n$  which may be risky (set in sections 6.2 and 7.1.1), and whether DDLab and your CPU are 32-bit or 64-bit. All possibilities of  $k_{Lim}$  are set out in table 7.1 for  $v=2$  to 8, including the size of the corresponding rule-tables,  $S$ , and this information is included in the prompts. There is a distinction in  $k_{Lim}$  between “basic” limits — relatively safe sizes, and extended limits “exLimits”.

Note that  $k_{Lim}$  does not exceed 27 because this is the maximum  $k$  in predefined neighborhood templates, described in chapter 10.1.

### 7.2.1 TFO-mode with high $v, k$

High values of  $v, k$  (values-range combined with neighborhood) as in table 7.1(b), can result in a longish delay while constructing the  $v, k$  kcode tree, required in TFO-mode irrespective of whether kcode or tcode is later selected.

While the kcode tree is being constructed (with no abort option) a top center panel monitors progress as a percentage.

**setting up kcode 95%** *(for example)*

If this delay occurs, it will be after the “WIRING” prompt in section 11.1.

<sup>1</sup>About 20min for the highest  $k_{Lim}$  values with “exLimits” active, on my ThinkPad laptop running ubuntu 11.10.

<sup>2</sup>Note that  $k_{Lim}$  is different from  $k_{max}$  which is a deliberate selection of the maximum  $k$  in a  $k$ -mix, a network with mixed neighborhood sizes,  $k_{max} \leq k_{Lim}$ .

applies to 32bit and 64bit

basic — <i>safe</i>					exLimits — <i>risky</i> > <i>safe(a)</i>		
FIELD/SEED			TFO kcode		TFO tcode		
$v$	$k_{Lim}$	$S$	$k_{Lim}$	$S$	$v$	$k_{Lim}$	$S$
2	13	8162	27	28	2	27	28
3	9	19683	27	406	3	27	406
4	7	16484	26	33654	4	27	4060
5	6	15629	25	23751	5	27	31465
6	5	16807	17	26334	6	27	201376
7	5	16807	13	27132	7	25	593775
8	4	4096	11	31824	8	20	888030

(a)

32-bit			64-bit		
exLimits — <i>risky</i>			exLimits — <i>risky</i>		
FIELD/SEED			FIELD/SEED		
$v$	$k_{Lim}$	$S$	$v$	$k_{Lim}$	$S$
2	23	8388607	2	27	134217727
3	14	4782968	3	19	1162261466
4	11	4194303	4	15	1073741823
5	9	1953124	5	13	1220703124
6	8	1679615	6	11	362797055
7	8	5764800	7	11	1977326742
8	7	2097151	8	10	1073741823

(c)

(d)

Table 7.1: The upper limits of  $k$  ( $k_{Lim}$ ) and the sizes  $S$  of rule-tables for different value-ranges  $v$ . FIELD/SEED-modes (rcode) are capable of computing basins of attraction or subtrees. TFO-mode (kcode or tcode) have smaller rule-tables so  $k_{Lim}$  can be larger. Other factors allowing greater but risky  $k_{Lim}$  are “exLimits” and whether DDLab and your CPU are 32-bit or 64-bit.

basic — 32-bit and 64-bit — <i>safe</i>				exLimits — 64-bit — 4.0 GB <i>risky</i>			
$v$	$n_{Lim}$	$v^n \approx$ billions	memory $\approx$ GB	$v$	$n_{Lim}$	$v^n \approx$ billions	memory $\approx$ GB
2	31	2.15	0.25	2	35	34.36	4.00
3	20	3.49	0.41	3	22	31.38	3.66
4	15	1.07	0.13	4	17	17.18	2.00
5	13	1.22	0.15	5	15	30.52	3.56
6	12	2.18	0.26	6	13	13.06	1.53
7	11	1.98	0.24	7	12	13.84	1.62
8	10	1.07	0.13	8	11	8.59	1.00

Table 7.2: For a basin of attraction field (FIELD-mode) the tables show the maximum network size,  $n_{Lim}$ , for different value-ranges  $v$ , the corresponding maximum state-space  $v^n$  in approximate billions, and the memory required in gigabytes (GB). *Left*: basic  $n_{Lim}$ . *Right*: increased  $n_{Lim}$  with exLimit set and default available memory of 4.0 GB — usually doubling memory increases  $n_{Lim}$  by one. These limits also apply to the maximum number of most significant bits/values for the state-space matrix (section 31.2.2.1).

---

### 7.3 Limits on network size for FIELD-mode, $n_{Lim}$

For basin of attraction fields (FIELD-mode) the upper limits<sup>3</sup> 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. To increase  $n_{Lim}$  beyond the basic settings (64-bit only), select “exLimits” in section 6.2, and set the available memory (RAM+SWAP) as described in section 6.2.4 — the default is 4.0 GB.

---

### 7.4 exLimits risks and insufficient memory

Extending  $k_{Lim}$  or  $n_{Lim}$  to high values with “exLimits” can be risky because it can test your available memory or your patience — attractor basins for large networks can take a long time. Various progress reports and warnings are provided, but if RAM+SWAP is exceeded for any reason DDLab usually exits gracefully with a top-left notice similar to the following,

```
insufficient memory for tickoff-FIELD (for example)
image: print/save-p/s, options-o
return to Exit:
```

If this fails for any reason, in Linux-like systems use “**top**” to “**kill**” the DDLab process.

---

### 7.5 Limits on network size — exhaustive algorithm, $n_{exhL}$

The exhaustive reverse algorithm (section 29.7) relies on computing the successor of every state in state-space — a list of  $v^n$  “exhaustive pairs”, so is highly sensitive to increasing network size  $n$  but independent of neighborhood size  $k$ , though the  $k_{Lim}$  in table 7.1 still applies. The network size limits  $n_{exhL}$  for the exhaustive algorithm for both FIELD-mode and SEED-mode (single basins and subtrees), are shown in table 29.1 and do not depend on “exLimits” (section 7.1.1). The exhaustive algorithm applies to random maps (section 29.8) and sequential/asynchronous updating (section 29.9), as well as providing an alternative reverse algorithm for CA, RBN and DDN.

---

<sup>3</sup>These limits are required so that the size of state-space  $v^n$  is within the maximum value of an unsigned long integer,  $2^{32}-1$  for a 32-bit CPU and  $2^{64}-1$  for a 64-bit CPU. This number is used to index bits in a char array that identifies “used” states, and this array must also fit within available memory: RAM+SWAP.

# 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 superseded 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 patterns.

The chapter also looks at the upper limits of  $n$  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

In FIELD-mode, it is possible to generate a series of basin of attraction fields for a range of 1d network sizes (figure 8.1), 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” [31]. In SEED-mode, 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:**

If **r** is selected, top-right prompts are presented to set the start and end size of the network,

**FIELD-mode range n: safe<=20 exceed with caution! nLimit=31**

**range size n: start (def 6): end (def 15):** (*for v=2*)

**SEED-mode range n: safe<=27 exceed with caution! nLimit=131071**

**range size n: start (def 13): end (def 18):** (*independent of v*)

For FIELD-mode defaults depend on the value-range  $v$ , in SEED-mode they are independent of  $v$ . Enter the start and end sizes, or enter **return** to accept the defaults, taking care to avoid excessively large sizes (section 8.3 below). Maximum network sizes in FIELD-mode are shown in table 7.2. In SEED-mode, large sizes<sup>1</sup> are only practical for “chain rule” subtrees (see 16.11) because of time/memory constraints.

---

<sup>1</sup>For 64-bit systems and exLimits set, nLimit=67108863 in SEED-mode

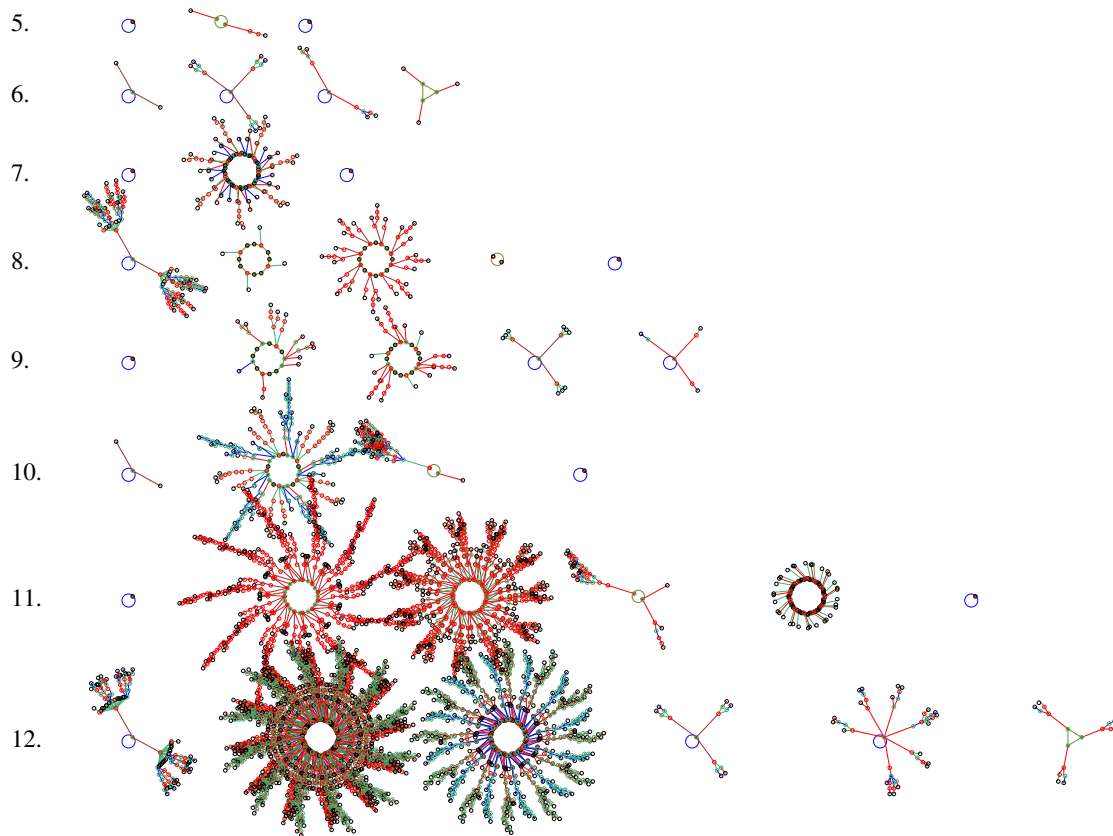


Figure 8.1: Basin of attraction fields for a range of network size 5-12.  $v2k4$  rcode(hex)=61a4.

---

## 8.2 Setting the size of one network, 1d

If a range of sizes was not selected in section 8.1, the size of a single 1d network is set with the next main sequence prompt — one of the following is presented, depending on the current mode — FIELD, SEED, or TFO (selected at the first prompt, section 6.2),

*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:**

Enter the required 1d network size,  $n$ . The 1d network size can also be set later as part of the wiring options (chapters 11 to 12), where 2d or 3d network sizes are set, in which case these prompts in sections 8.1 – 8.2 are superseded and may be ignored with **return**.

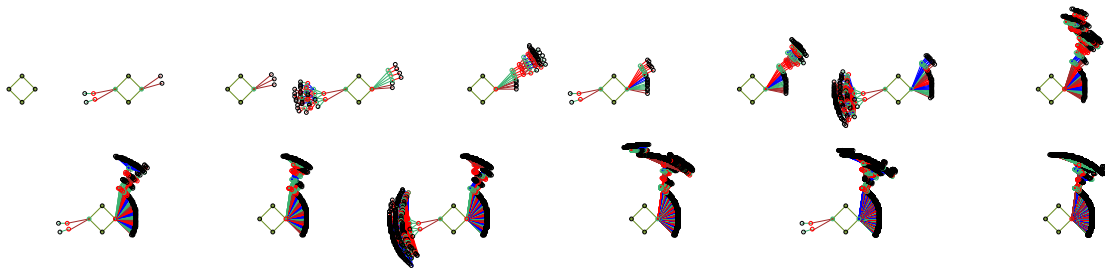


Figure 8.2: Single basins for a range of network sizes containing the state all 0s,  $n=1$  to 15,  $v4k2$   $\text{rcode(hex)}=\text{a7857c0d}$ .

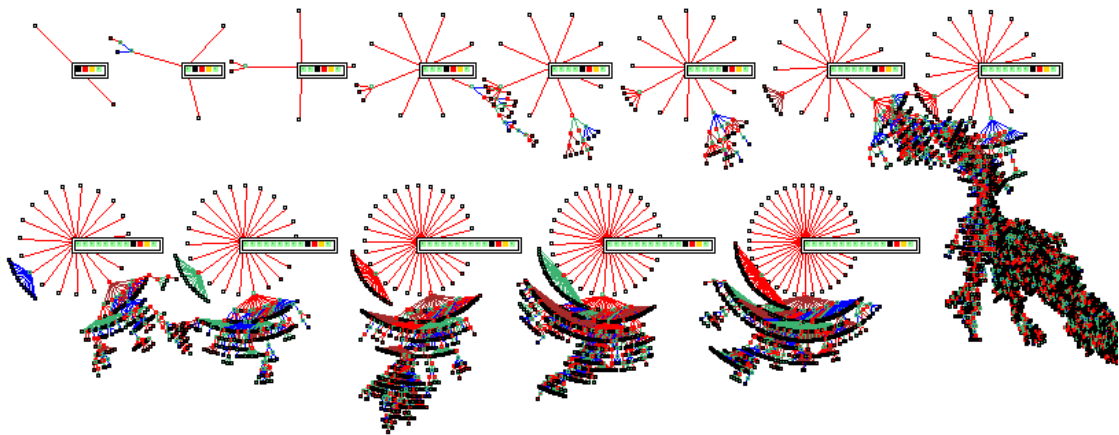


Figure 8.3: Subtrees for a range of network sizes from a root state containing from the string 3210, which is highlighted,  $n=4$  to 15,  $v4k3$   $\text{rcode(hex)}=\text{1c49a5b05dbcdd148377635fb0b60d84}$ .

At the same time that the network size main sequence prompts are presented, a top-right warning is displayed as follows,

*for FIELD-mode*

**FIELD-mode: network size n: safe<=20, nLimit=31**

**exceed with caution!** (for  $v=2$ , for limits in general see section 7.3)

*for SEED/TFO-modes — SEED-mode shown*

**SEED-mode: network size n: safe<=65535, nLimit=8388607** (or 4294967295)

**exceed with caution!** can be reset in WIRING for 1/2/3d, rnd/special

The network size  $n$  cannot be greater than  $n_{Lim}$ , described in section 8.3 below. If a larger  $n$  is selected, the default  $n$  is applied with a message similar to the following,

... **:33 too big! n=10** (for FIELD-mode,  $n_{Lim}$  is listed in table 7.2)

... **:8400000 too big! limit=8388607! n=14** (for SEED/TFO-mode)

---

### 8.3 Network size limits

Network size limits are also discussed in sections 1.6.1 and for FIELD-mode they are listed in section 7.3 for values-range  $v$ . For SEED/TFO-modes  $n_{Lim}$  is independent of  $v$ . To summarize, the upper limit of network size,  $n_{Lim}$ , supported by DDLab is as follows,

*for FIELD-mode*

**Basin of attraction fields:**  $n_{Lim}$  is set out in table 7.2, and varies according to value-range  $v$  (from 2 to 8), and if “exLimits” is set in a 64-bit CPU (section 6.2.4). For  $v=2$  and basic limits,  $n_{Lim}=31$  (figure 8.6) and for  $v=8$   $n_{Lim}=10$ . In practice  $n$  should be well below these maximum limits.

*for SEED-mode running backwards*

**Single basins and subtrees:**  $n_{Lim}=8388607$ , or 4294967295 if “exLimits” is set in a 64-bit CPU (section 6.2.4). In practice  $n$  should be very much below these maximum limits. For a single basin try  $n=18$ , for a subtree  $n \leq 50$  — but for a practical size try  $n=25$ . However, the subtrees of “chain rules” (section 16.11) can be run backwards for extremely large  $n$ , as in figure 8.4.

*for SEED-mode running forwards, and TFO-mode*

**Space-time patterns (1d, 2d or 3d):**  $n_{Lim}=8388607$ , or 4294967295 if “exLimits” is set in a 64-bit CPU (section 6.2.4). For 2d  $i, j$  or 3d  $i, j, h$  any sub-multiples of  $n$  apply: giving a square with sides 2896 (or 65535), a cube with sides 203 (or 1625). For reasons of speed and memory, much smaller sizes are appropriate. For 1d networks shown in 1d, a preferable size is  $n=150$  to 200, to fit the screen and for a clear view of analytical graphic windows, but to see these windows for larger 1d networks, they can be shown in 2d or 3d.

---

### 8.4 Computational and speed limitations for attractor basins

In general, large networks (size  $n$ ) may be run “forwards” to generate space-time patterns, and also find attractors for rules with a low  $Z$ -parameter [38], 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 local 1d CA.

It is not only  $n$  that effects the 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 (nonlocal) 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 [38] which varies between 0 and 1, or the average  $Z$  across the network for



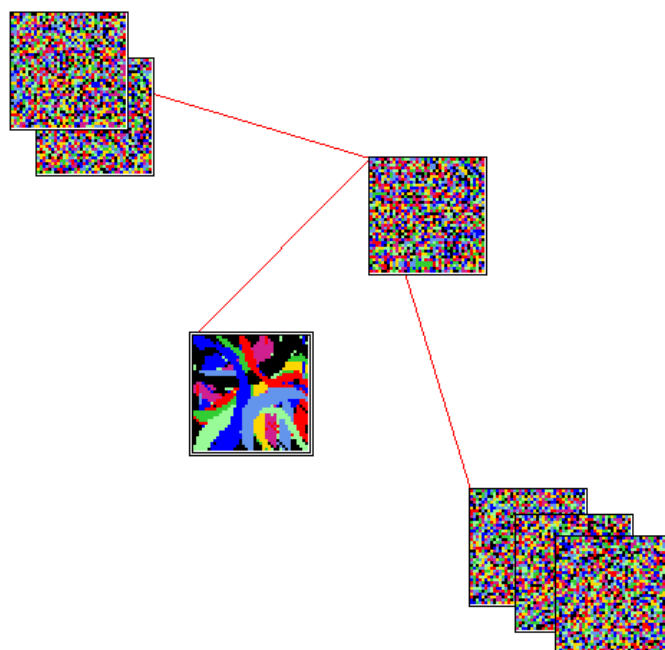


Figure 8.4: A subtree for a 1d CA chain-rule for a large network.  $n=1600$  shown as  $40 \times 40$ ,  $v8k4$ . The root state of the subtree is located left of center.

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.

### 8.4.1 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<sup>2</sup> 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 27.2).

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 may impose unacceptable time, memory and display constraints. Single basins, and especially subtrees, 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.

<sup>2</sup>Lenovo 3000 N100 Laptop, Duo processor T2300e 1.66GHz, 1024 Mb RAM, running Linux Ubuntu 6.06.

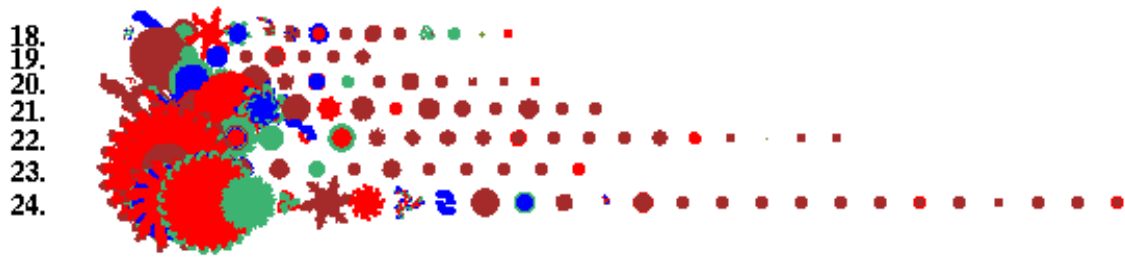


Figure 8.5: Basin of attraction fields for a range of network sizes,  $v2k5$ ,  $n=18$  to 24, to assemble part of the data for table 8.1 below.

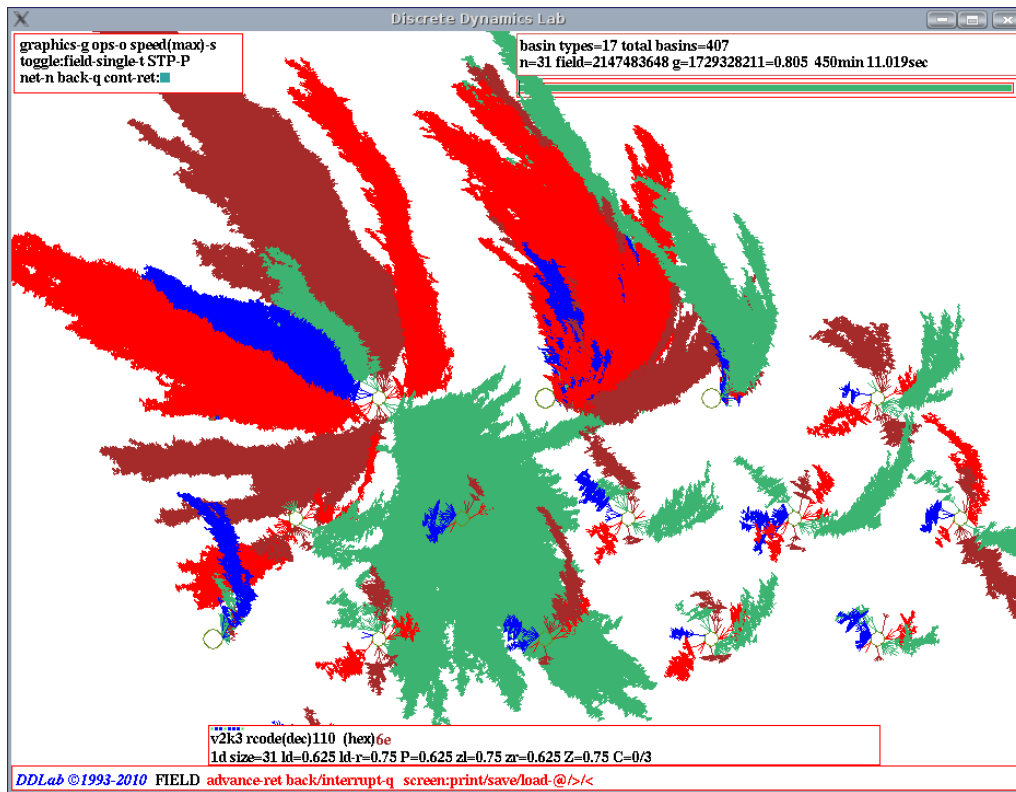


Figure 8.6: The basin of attraction field of  $v2k3$  rcode (dec)110, for  $n=31$ , which is the usual  $n_{Lim}$  for  $v=2$ , elapsed times to generate 7.5 hours. The field is shown on the DDLab screen, with compression on (section 26.2), equivalent trees and subtrees suppressed (section 26.2.3), and nodes suppressed (section 26.3).

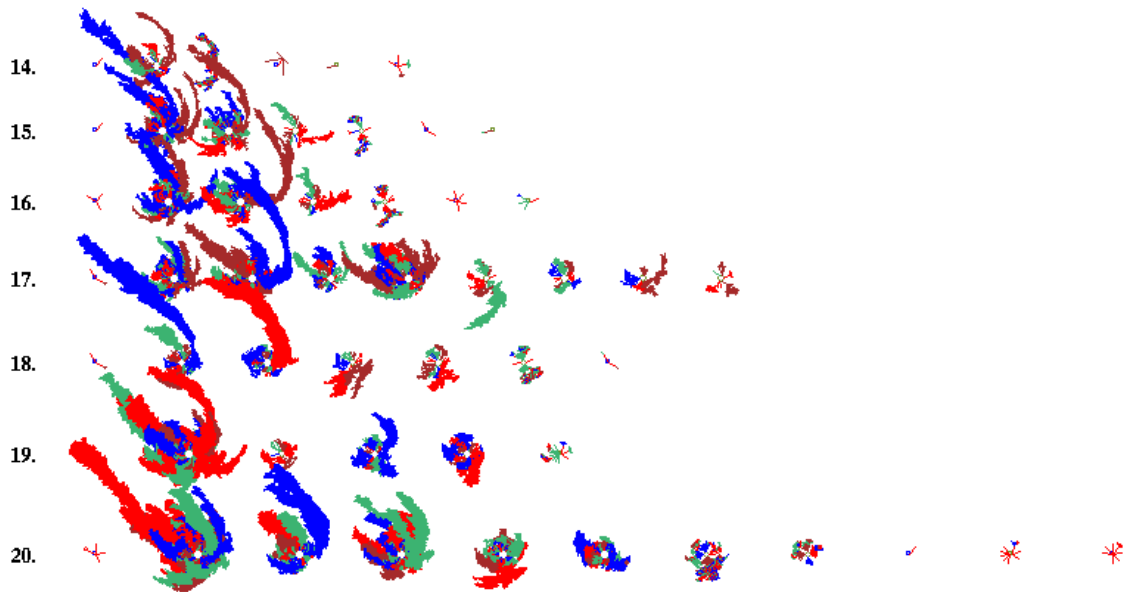


Figure 8.7: Basin of attraction fields for a range of network sizes, RBN,  $v2k3$ ,  $n=14$  to 20, to assemble part of the data for table 8.3 below.

### 8.4.2 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 filenames for  $v \geq 3$ .

$n$	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.

### 8.4.3 Examples for RBN and DDN

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

The same rules were used as for the CA in section 8.4.2, 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.

$n$	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.

# Chapter 9

## Neighborhood, $k$ , or mixed- $k$

This chapter describes setting a homogeneous 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 nonlocal), setting 1d, 2d or 3d wiring, and other special wiring options come at a later stage (chapters 11, 12) — in that any 1d  $k$  entries in this chapter will be superseded, 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 patterns.

---

### 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, where the default  $k$  and  $k_{Lim}$  depend on TFO-mode (section 6.1), value-range  $v$  (section 7.1), and “exLimit” (sections 6.2.4 and 7.1.1) as previously set,

**Neighborhood size k: kmix-m, or enter 1-13 (def 3):** *(for example)*

Enter the required value of  $k$  (or **m** for a  $k$ -mix). A valid  $k$  set here becomes the new default.

If the  $k$  entered is larger than  $k_{Lim}$  (section 7.2) the default  $k$  is applied with the following message,

... **:33 too big! k=3** *(for example)*

If **m** is entered for a  $k$ -mix, the actual mix will be set in section 9.3 below.

At the same time that the “Neighborhood” prompt above, a top-right notice is displayed as follows,

**k and kmix can be reset in WIRING for 1/2/3d, rnd/special**

This is a reminder that  $k$ , the  $k$ -mix, the network wiring for 1d, 2d, and 3d, and other special features can be reset in subsequent wiring options (chapters 11, 12), so the main sequence prompt can be ignored by entering **return**.

Once the network’s rule or rules have been set (chapters 13 to 16), the network’s  $k$  or  $k$ -mix may be “neutrally” modified, both by increasing  $k$  or reducing to effective  $k$  (sections 18.7.1—18.7.3), for the network as a whole or for a particular cell.

### 9.1.1 $k_{Lim}$ and minimum $k$

The maximum number of input wires to a cell,  $k_{Lim}$ , is constrained by increasing value-range  $v$  (section 7.2) because rule-tables get longer, bearing in mind that rcode (FIELD/SEED-mode) has longer rule-tables than kcode (TFO-mode). All possibilities of  $k_{Lim}$  are set out in table 7.1 and depend also on your CPU and DDLab version, 32-bit or 64-bit, and if “exLimit” is active (section 6.2.4) and if so whether  $k_{Lim}$  was extended (section 7.1.1). For example, for basic rcode  $k_{Lim}=13$  for  $v=2$  and  $k_{Lim}=4$  for  $v=8$ , but for 64-bits and exLimit applied  $k_{Lim}=27$  for  $v=2$  and  $k_{Lim}=10$  for  $v=8$ , though these higher limits can test your available memory or your patience (section 7.4). For basic kcode  $k_{Lim}=27$  for  $v=2$  and  $k_{Lim}=10$  for  $v=8$ . Note that  $k_{Lim}$  is different from  $k_{max}$  which is a deliberate selection of the maximum  $k$  in a  $k$ -mix,  $k_{max} \leq k_{Lim}$ .

---

## 9.2 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	hex	dec
2	10	02	2
3	210	24	21
4	3210	e4	228
5	43210	4688	2930
6	543210	02c688	44790
7	6543210	1ac688	800667
8	76543210	fac688	16434824

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. The rules can be entered in three alternative ways — as a bit/value rule-table, as the equivalent in hexadecimal (hex), and in decimal (dec).

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 receiving any inputs from other cells, and that it will conserve its current value. However, other cells may receive inputs from the effective  $k=0$  cell. DDLab can identify any self-wired  $k=1$  cells in a mixed- $k$  network, and gives an option to set the rule at these cells to make them effectively  $k=0$  automatically (section 14.3), or this can be done by hand — the easiest way is to “kill a cell” from the 1d wiring graphic (section 17.9.9). Setting effective  $k=0$  is useful for providing a fixed input signal to a network, or potentially for an arbitrary external input.

---

## 9.3 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 (chapter 17).

---

## 9.4 Loading a $k$ -mix file

Enter **l** in section 9.3 above to load the  $k$ -mix as a .mix file (section 35.3). The file would have been saved in sections 19.5 or 9.11.3. 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 top-right 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 *Network filing options* — section 19.1 (see also section 17.9.12).

---

## 9.5 Setting the $k$ -mix by hand

If **h** is selected in section 9.3, 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.

---

## 9.6 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 9.11.

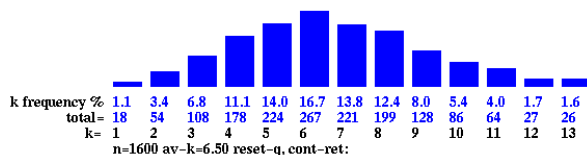


Figure 9.1: An example of a normal (Poisson) distribution of the frequencies of different  $k$ .  $n=40 \times 40$ , the range of  $k=13$  and the average  $k=6.5$

## 9.7 Specify each $k$ , or power-law distribution

Enter **s** in section 9.3, 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 9.7.1 to 9.7.3 below.

### 9.7.1 Specify the percentage of different $k$

If **return** is entered in section 9.7 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). Alternatively, 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 9.3. At any stage from  $k=2$  onwards it is possible to backtrack and revise with **b**.

### 9.7.2 Setting a power-law distribution of $k$

Enter **p** at the prompt in section 9.7 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 target 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 (section 17.9.5, figure 17.22).

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 [4] to the world-wide-web. The graph of a “scale-free” network is shown in figure 20.2.



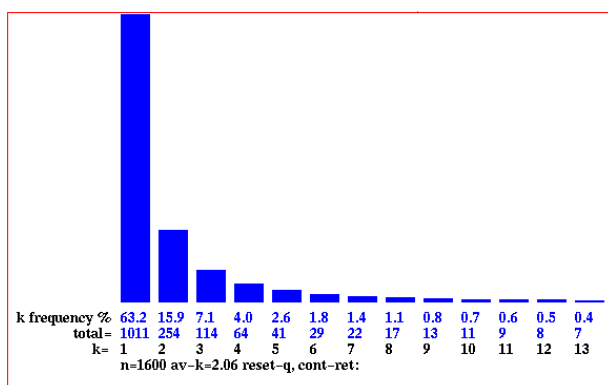


Figure 9.2: An example of a power-law distribution of the frequencies of different  $k$ . In this case  $n=40 \times 40$ , the range of  $k=13$ , the power-law exponent is 2, and the average  $k=2.05$ .

### 9.7.3 Showing the distribution

When the  $k$ -distribution has been assigned, either by specifying the percentage of different  $k$  in section 9.7.1, or by setting a power-law in section 9.7.2, data on the distribution, and the following prompt, are presented in a top-right window (for example, for a 2d  $40 \times 40$  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):
```

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 9.11 (and later in section 17.9.13).

## 9.8 Setting the $k$ -mix at random

If **return** was entered in section 9.3, 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 referred to as  $k_{max}$ . The different  $k$ 's will be assigned with equal probability.

## 9.9 Setting a $k$ -mix with uniform $k$

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

A  $k$ -mix with homogeneous- $k$  is easily created in section 9.7.1, by setting the percentage of the required  $k$  at 100%, or in section 9.8 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 nonlocal wiring and mixed rules even if the wiring is set as local 1d (section 12.4.1), and there is just one rule (section 14.4.3). These issues relate to how memory is allocated to different types of network.

---

## 9.10 Increasing $k_{max}$

Options for changing  $k$  for individual cells (or blocks of cells) occur at later stages in DDLab (sections 17.9.8, 18.7.1).

The default  $k_{max}$  is the upper bound set previously, for example in section 9.8. 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 subsequently increased up the new  $k_{max}$ . For example, if 100%  $k=5$  is set in section 9.7.1 (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  $\leq 9$ .

---

## 9.11 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 TFO-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.

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:
```

```
k-mix 1-25, maxk=25, net size=200, startindex=199
2522152313241932153168811416217242418241251618218271521022495215
3151423162396241012233112214162112423182481123341314625151472323
2016252413211111822824135136212314214231716181511201421217323112
31891238255516241915212442211617231551619227192125215285102520510
2762512522510513101718221017245211910187941518242319252111182311
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  $k=1-25$ ,  $n=200$  (TBO-mode). To distinguish each (decimal) number, colors alternate between black and red.

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 9.3 and reset the  $k$ -mix. Enter **s** to save the  $k$ -mix in a `.mix` file (section 35.3) which can be loaded as in section 9.4. Enter **return** to accept the  $k$ -mix. The other options are explained below.

### 9.11.1 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 9.11) 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:
```

### 9.11.2 Jumping to a new cell index

Alternatively, enter **j** in section 9.11.1 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)
```

### 9.11.3 Saving the $k$ -mix file

Enter **s** in section 9.11 to save the  $k$ -mix. A filing prompt box will allow saving a **.mix** file — the default filename is **mymix.mix** (section 35.3). The  $k$ -mix can also be saved in section 19.5, but can only be loaded in section 9.4. However, the  $k$ -mix in mixed- $k$  networks is implicit in the **.w\_s**, **.r\_s** and **.wrs** files (section 19.2), so does not usually need to be saved separately.

## 9.12 Reviewing the $k$ -mix within “network architecture”

The  $k$ -mix may also be reviewed in *Reviewing network architecture* as described (chapter 17), where options allow changing  $k$  (including “neutral”  $k$  changes) up to  $k_{max}$  set in section 9.10, 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.

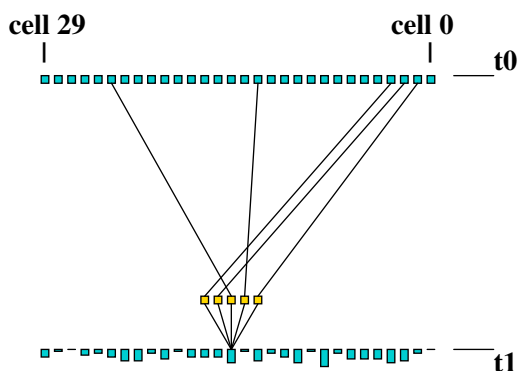


Figure 9.4: The 1d wiring graphic for a  $k$ -mix, showing the wiring between successive time-steps,  $n=30$ . Each cell’s “outwires” are represented by the heights at time-step  $t_1$ .

# 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 its 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, 2d and 3d the neighborhoods are defined in DDLab for  $k= 1$  to 27 ( $k \geq 24$  best in TFO-mode).

The following sections describe the predefined neighborhoods and how they are indexed. Note that in 2d the neighborhoods can be square or hexagonal (sometimes either) to run on a square or hexagonal lattice, as shown in figure 10.2. It is also possible to construct any arbitrary neighborhood and assign it uniformly throughout the network (section 12.5.11).

#### 10.1.1 The pseudo-neighborhood, RBN and DDN

For networks with nonlocal (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 CA, RBN and DDN (chapter 13).

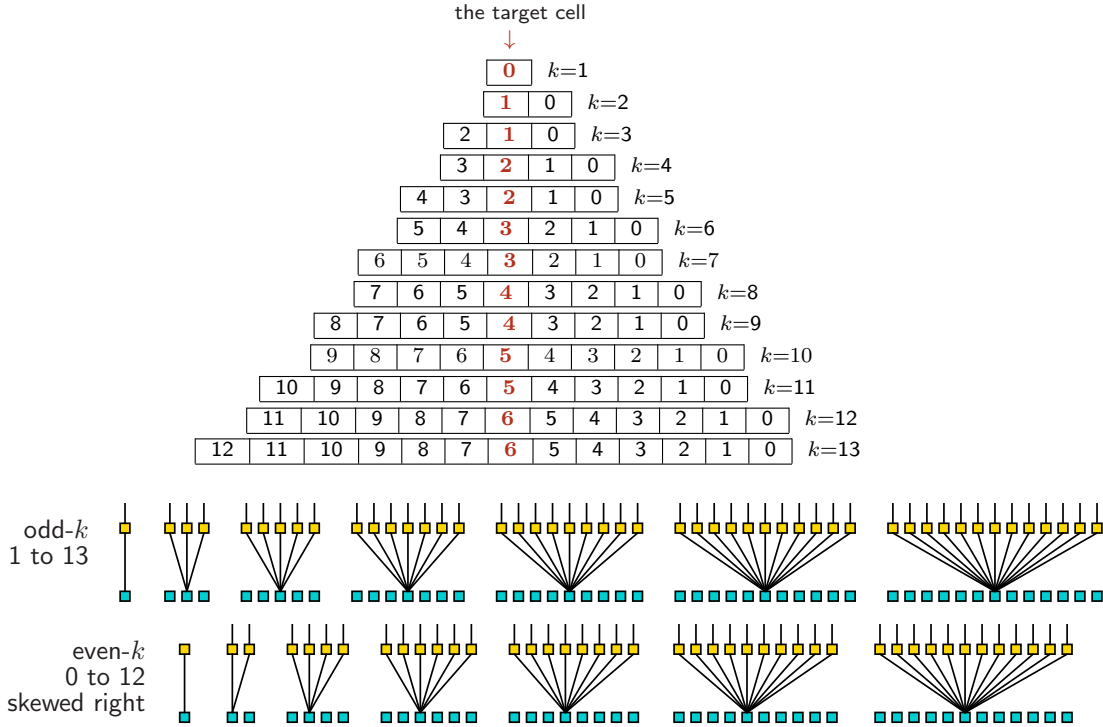


Figure 10.1: 1d neighborhood templates as defined in DDLab, shown from  $k=1$  to 13. *Top*: showing indexing. *Bottom*: graphically as in section 17.6. The same principle applies for  $k=14$  to 27. Note that for even  $k$  the neighborhood is asymmetric, skewed to the right. For effective  $k=0$  see section 9.2.

### 10.1.2 1d neighborhood

The 1d neighborhoods for  $k=1$  to 27 are defined and indexed according to figure 10.1. Cells in the neighborhoods are indexed in reverse order,  $k-1, k-2, \dots, 0$ . For odd  $k$  the target cell is centered, for even  $k$  the neighborhood is asymmetric with an extra cell on the right.

### 10.1.3 2d neighborhood

The 2d neighborhoods templates for  $k=2$  to 27 are shown in figure 10.2 ( $k=1$  is self evident). Each  $k$  has a template based on either a square or hexagonal lattice, sometimes both apply, designed for best symmetry. If both, those shown in a frame need to be selected (section 11.1 or 12.1). Where a square neighborhood is missing in the definitions, a distorted hex neighborhood will apply in a square layout, and vice-versa. The layout can be toggled between square and hex in the 2d wiring graphic (section 17.7) and on-the-fly (section 32.9.2) but this does not change the underlying templates as originally defined. To achieve maximum symmetry and for outer-totalistic rules (section 14.2) the target cell itself is sometimes not included in the neighborhood template. Cells in the neighborhood are indexed starting with 0 then from right to left in ascending rows, as in the example for  $k=9$  in the bottom-right corner of figure 10.2. Note that for a hexagonal lattice to work properly, the height of the network,  $j$ , should be even.

With a square layout and template,  $k=5$  is known as the von Neuman neighborhood, and  $k=9$  is the Moore neighborhood — applied in the “game-of-Life” (section 16.10). With a hex layout and template,  $k=6$  applies to the “beehive rule” [44] and  $k=7$  to the “spiral rule” [45].

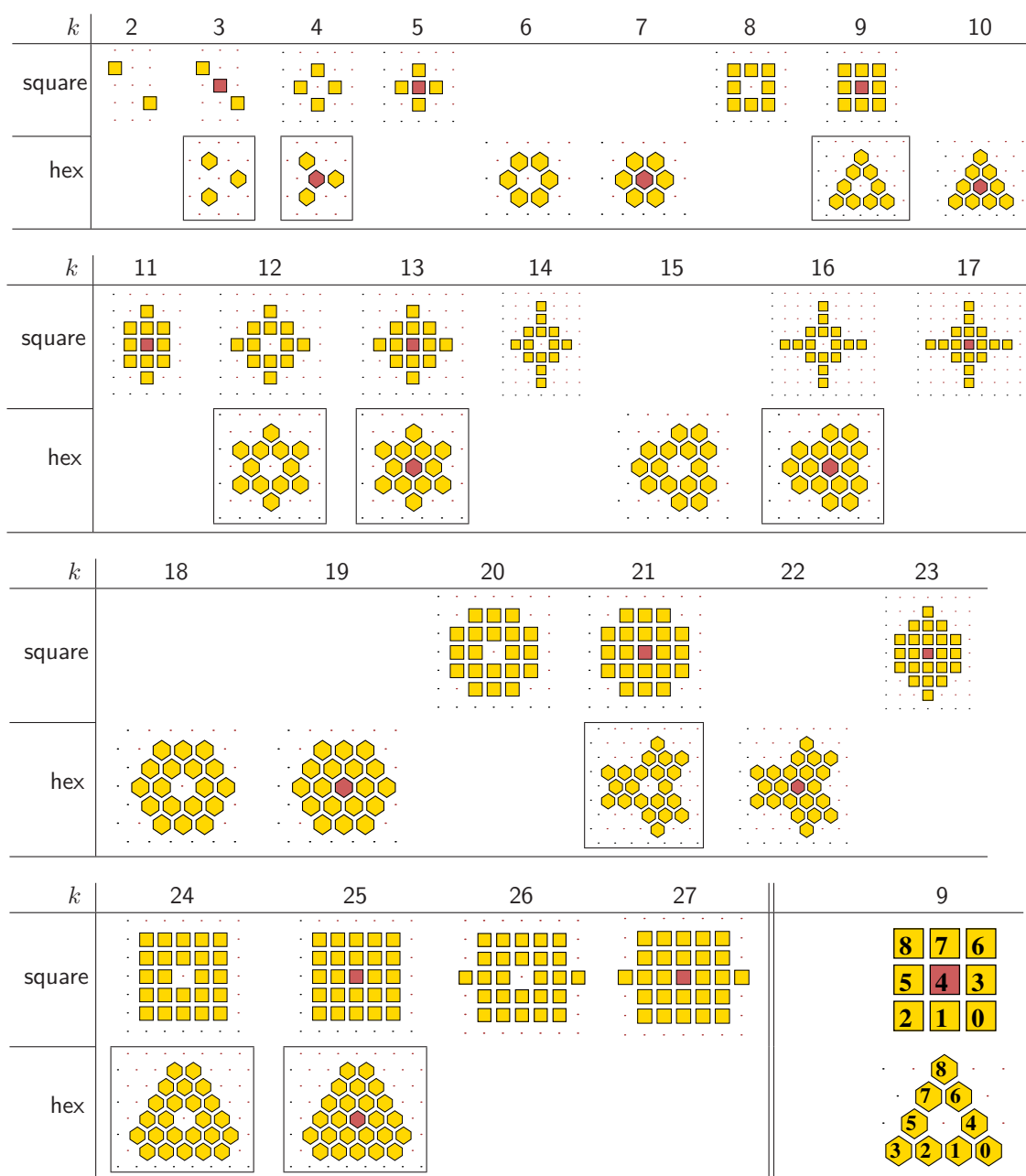


Figure 10.2: 2d neighborhood templates as defined in DDLab, for  $k=2$  to 27, square, hex or both, designed for best symmetry. The target cell may be excluded and is otherwise shown red.

*Bottom-Right:* templates are indexed starting with 0 then from right to left in ascending rows, as in these examples for  $k=9$  for square and hexagonal neighborhoods. The index numbering is toggled on/off with *index-i* (section 17.4).

### 10.1.4 3d neighborhood

3d neighborhoods for  $k=2$  to 27 are defined in figure 10.3 for a cubic grid on the 3-torus. For even  $k$  the target cell itself is not included in the neighborhood.

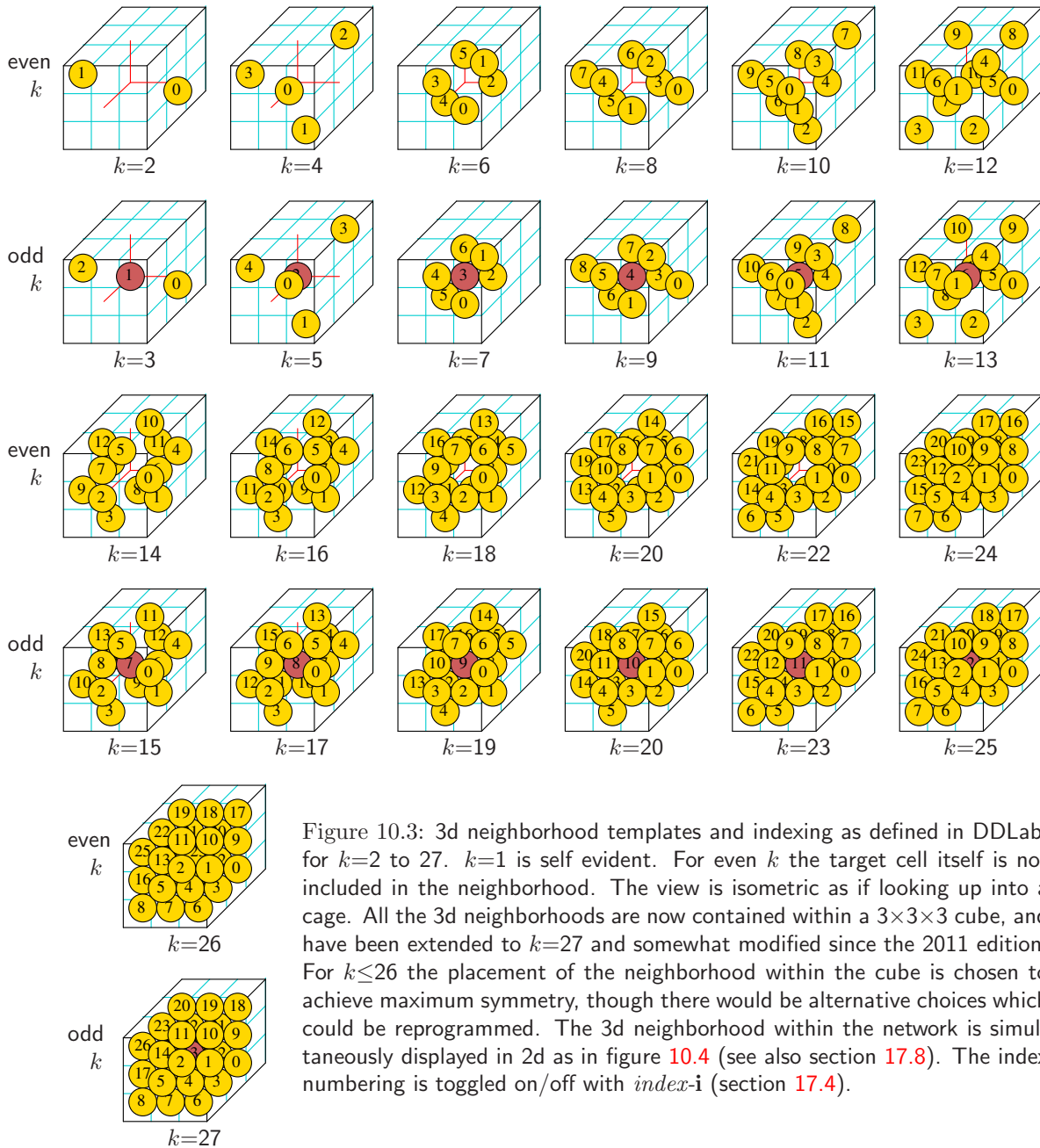


Figure 10.3: 3d neighborhood templates and indexing as defined in DDLab, for  $k=2$  to 27.  $k=1$  is self evident. For even  $k$  the target cell itself is not included in the neighborhood. The view is isometric as if looking up into a cage. All the 3d neighborhoods are now contained within a  $3 \times 3 \times 3$  cube, and have been extended to  $k=27$  and somewhat modified since the 2011 edition. For  $k \leq 26$  the placement of the neighborhood within the cube is chosen to achieve maximum symmetry, though there would be alternative choices which could be reprogrammed. The 3d neighborhood within the network is simultaneously displayed in 2d as in figure 10.4 (see also section 17.8). The index numbering is toggled on/off with *index-i* (section 17.4).

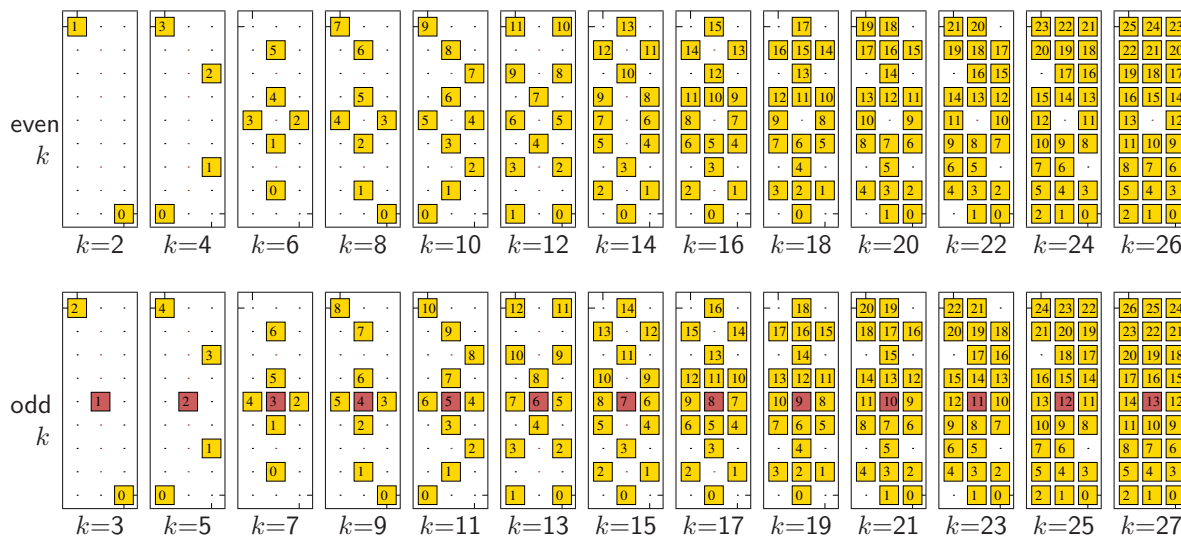


Figure 10.4: 3d neighborhood templates are simultaneously displayed in 2d as well as 3d (section 17.8). These 2d graphics for  $k=2$  to 27 show the 3 levels of each  $3 \times 3 \times 3$  cube, one below the other, representing the neighborhood templates in figure 10.3. The index numbers start with 0 then from right to left in ascending rows. The index numbering is toggled on/off with *index-i* (section 17.4).

## 10.2 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 10.5, 10.6 and 10.7 — examples made with the network-graph functions 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 target cell itself. For networks with fully random wiring the geometry simply allows convenient indexing and representation. The cells are indexed  $n-1, \dots, 0$ , or according to their I,J,H coordinates.

CA usually have periodic boundary conditions, where opposite edges join, so CA (local) wiring in 1d creates a ring of cells, in 2d a regular square or hexagonal lattice on the torus, in 3d a 3-torus. By default, the neighborhood templates of CA, RBN and DDN operate within these periodic boundaries, but this can be changed to Null boundaries (section 2.7) — for attractor basins in section 26.1 — for space-time patterns in section 31.3 or on-the-fly in section 32.7.2. Space-time patterns can also be set with fixed borders in section 32.16.6.



### 10.2.1 1d network indexing

1d networks, size  $n$ , are indexed  $n-1, \dots, 0$ , where  $i = n$ , and are usually laid out horizontally following figure 10.5 *Left*.

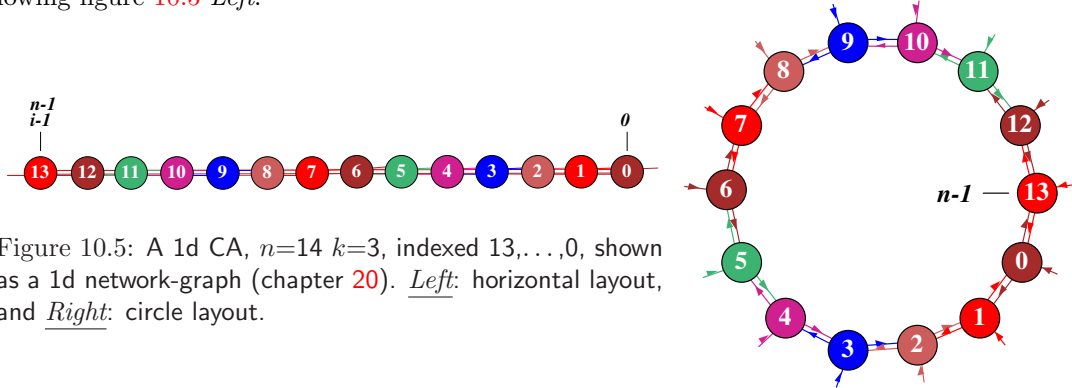


Figure 10.5: A 1d CA,  $n=14$   $k=3$ , indexed  $13, \dots, 0$ , shown as a 1d network-graph (chapter 20). *Left*: horizontal layout, and *Right*: circle layout.

### 10.2.2 2d network indexing

2d networks, size  $n$ , are indexed  $n-1, \dots, 0$ , with either square or hex layout as in figure 10.6.  $i$  is the width (columns),  $j$  is the depth (rows). Note that the cell indexes shown,  $x$ , are 1d indexes.

For a 2d network, size  $i, j$ , to convert the coordinates of a cell,  $I, J$ , to a 1d index,  $x = iJ + I$ . Conversely,  $I = x \bmod i$ ,  $J = \lfloor \frac{x}{i} \rfloor$ .

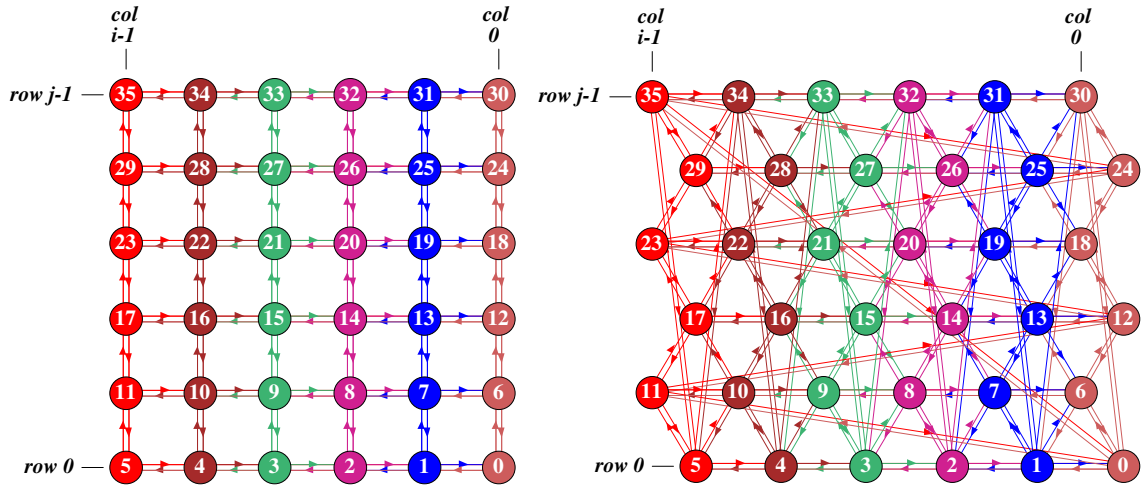


Figure 10.6: A 2d CA,  $n=36$ , where  $i \times j = 6 \times 6$ , indexed  $35, \dots, 0$ , shown as a 2d network-graph (chapter 20). *Left*: square layout ( $k=4$ ). *Right*: hex layout ( $k=6$ ). In the network-graph prompt (section 20.2.1) key  $2d(tog)$ -2 toggles between square and hex layout.

### 10.2.3 3d network indexing

3d networks size  $n$  are indexed  $n-1, \dots, 0$ , as in figure 10.7.  $i$  is the width (columns),  $j$  is the depth (rows), and  $h$  is the height (levels). Note that cell indexes shown,  $x$ , are 1d indexes, For a 3d network, size  $i, j, h$ , to convert the coordinates of a cell,  $I, J, H$ , to a 1d index,  $x = ijH + iJ + I$ . Conversely,  $I = x \bmod i$ ,  $J = \lfloor \frac{x \bmod ij}{i} \rfloor$ ,  $H = \lfloor \frac{x}{ij} \rfloor$ .

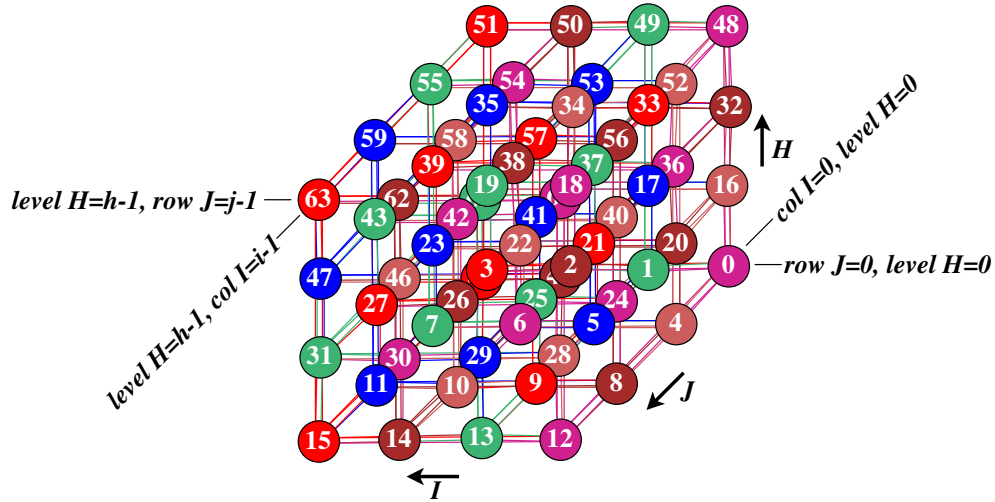


Figure 10.7: A 3d CA,  $n=64$ ,  $[i, j, h]=[4, 4, 4]$ ,  $k=6$ , indexed  $63, \dots, 0$ , shown as a 3d network-graph, (chapter 20). The graph should be viewed as if looking up from below into a cage, were the bottom layer of cells are numbered 0-15.

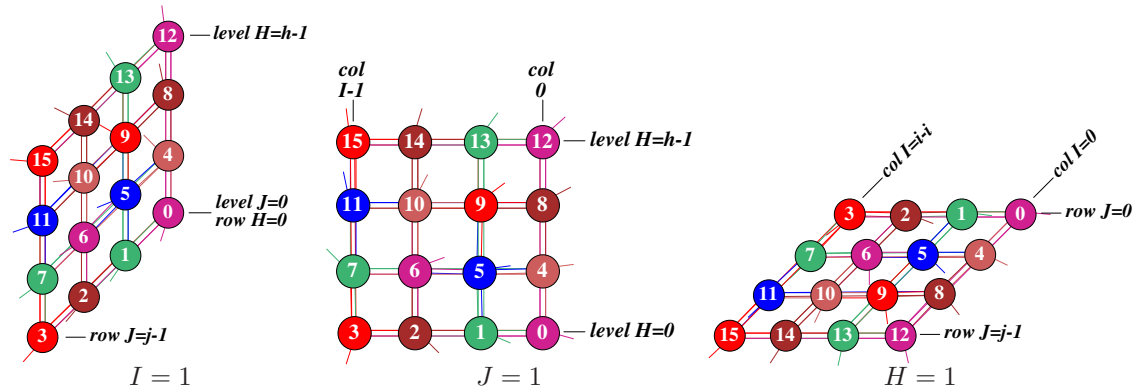


Figure 10.8: To clarify 3d network indexing, the network-graphs above show three examples where the width (columns)  $i$ , depth (rows)  $j$ , and height (levels)  $h$ , are made to equal 1 in turn, so from left to right  $[i, j, h]=[1, 4, 4], [4, 1, 4], [4, 4, 1]$ .  $k=6$ , cells numbered 0-15.

# 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. 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 Local 1d wiring

Enter **return** (the default) in section 11.1 above to set wiring as local 1d with periodic boundary conditions<sup>1</sup> and skip remaining wiring options. The network may be a CA, or may have mixed  $k$  as previously selected in chapter 9. The 1d network size would have been set in the main sequence prompt section 8.2.

---

### 11.3 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.

---

<sup>1</sup>Periodic boundary conditions can be reset to null boundary conditions (NBC, section 2.7) at later stages (sections 26.1, 31.3, 32.7.2).

## 11.4 Loading the wiring scheme

If **l** is selected in section 11.1, filing prompts (section 35.3) will allow a wiring scheme (**.w.s**) file to be loaded, provided the file is compatible with the base network, in that  $\text{file-}n \leq \text{base-}n$ , and  $\text{file } k_{max} \leq \text{base } k_{max}$  (section 19.4.1).

## 11.5 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).

## 11.6 2d or 3d wiring

If **2** or **3** is selected in section 11.1, the wiring will be set as local 2d or 3d CA wiring according to the default neighborhoods in sections 10.1.3 and 10.1.4. These options also apply for random 2d and 3d wiring set in section 12.3.1. In each case boundary conditions are periodic, where the 2d array can be imagined as drawn on the surface of a torus, and the 3d array on a 3-torus, though this can be overridden with null boundary conditions (NBC, 2.7). The network can also have a  $k$ -mix with CA wiring for each (different size) neighborhood. Entering **2x** or **2s** will force the neighborhood, and the initial presentation of the lattice, to be hex or square (orthogonal), overriding the defaults in section 10.1.3. Just the hex/square presentation can be toggled later in the program.

As well as setting local wiring, further prompts below will set 2d or 3d network size, and  $k$  or a  $k$ -mix. Previous  $n$  and  $k$  settings from the main sequence of prompts (sections 8.2 and 9.1) will be superseded.

### 11.6.1 Setting 2d and 3d network size — SEED/TFO-mode

For SEED-mode or TFO-mode, the following options set the  $i \times j$  (width $\times$ depth) dimensions of 2d networks, and the  $i \times j \times h$  (width $\times$ depth $\times$ height) dimensions of 3d networks,

*basic 2d networks*

**2d: square edges, safe $\leq$ 255, max=2896 incr with caution!**

**enter i (def 40): j (def 40):** (*for exLimits, max=65535*)

*basic 3d networks*

**3d: cube edges, safe $\leq$ 40, max=203 incr with caution!**

**enter i (def 9): j (def 9): h (def 9)** (*for exLimits, max 1625*)

Enter the width **i**, depth **j** (and height **h**) — the prompts are presented in turn — subsequent defaults may depend on initial sizes, keeping in mind the network size limits  $n_{Lim}$  (section 8.3); **safe** and **max** show the safe and maximum edges of a square or cube, but the edges can be any length making up a rectangle or cuboid, as long as  $i \times j$  or  $i \times j \times h$  is less than  $n_{Lim}=8388607$ , or 4294967295 if “exLimits” is set in a 64-bit CPU (section 6.2.4) — in this case the max square edge=65535, and the max cube edge=1625.

If the dimensions selected for a 2d or 3d network exceed  $n_{Lim}$  a message such as the following is displayed within the top-right window,

**2896x3000 too big! max size=8388607 cont-ret:** (for basic 2d)  
 or for 3d  
**203x203x300 too big! max size=8388607 cont-ret:** (for basic 3d)

File encoding for 2d and 3d networks treats the axes or edge dimensions differently for a “small” network  $n \leq 65534$  and a “big” network  $n \geq 65535$  (sections 19.3 and 21.9). For “small”  $n$  edges must not exceed 255. For “big”  $n$  the edge limit is 65535, though this would only be relevant in the extreme case where the other edges sizes were 1. Whether the network is “small” or “big” depends on the edge sizes selected. For example,  $i \times j$  of  $255 \times 256$  would not be allowed with the warning below, whereas  $255 \times 255$  (“small”  $n$ ) or  $256 \times 256$  (“big”  $n$ ) are acceptable. The following warnings might be displayed,

**255x256 &  $n \leq 65534$  file conflict! change i,j cont-ret:** (for a “small” network)  
**1x65536 2d edge max 65535 exceeded! change i,j cont-ret:** (for a “big” network)  
 or for 3d  
**1x1x256 &  $n \leq 65534$  file conflict! change i,j,h cont-ret:** (for a “small” 3d network)

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

## 11.6.2 Setting 2d and 3d network size — FIELD-mode

For FIELD-mode the prompts to set the size of 2d and 3d networks are similar to SEED/TFO-mode (section 11.6.1) except that the network size limits (section 8.3) are very much smaller and depend on the value-range  $v$ , as described in section 7.3.

The following options set the  $i \times j$  (width $\times$ depth) dimensions of 2d networks, and the  $i \times j \times h$  (width $\times$ depth $\times$ height) dimensions of 3d networks,

*2d networks*

**2d: safe square edges $\leq$ 5, nLimit=31 inc with caution!**  
**enter i (def 4): j (def 4):** ( $v=2$ , basic)

*3d networks*

**3d: safe cube edges $\leq$ 3, nLimit=35 incr with caution!**  
**enter i (def 3): j (def 3): h (def 3)** ( $v=2$ , exLimits)

Enter the width **i**, depth **j** (and height **h**) — the prompts are presented in turn — subsequent defaults may depend on initial sizes. **safe...** shows the safe edges of a square or cube, but the edges can be any length making up a rectangle or cuboid as long as the network size  $i \times j$  or  $i \times j \times h$  is less than  $n_{Lim}$ . For  $v=2$   $n_{Lim}=31$ , or 35 if “exLimits” is set in a 64-bit CPU (section 6.2.4). If the dimensions selected for a 2d or 3d network exceed  $n_{Lim}$  a message such as the following is displayed within the top-right window,

**5x7 too big! max size=31 cont-ret:** ( $v=2$ , 2d, basic)  
 or  
**3x3x4 too big! max size=35 cont-ret:** ( $v=2$ , 3d, exLimits)

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

## 11.7 Set $k$ , or $k$ -mix

If 2d or 3d was selected in section 11.1, the next top-right prompt resets  $k$ , or the  $k$ -mix, as described in chapter 9.

**Neighborhood size k: kmix-m, or enter 1-27 (def 5):** *(for example)*

The upper bound,  $k_{Lim}$ , depends on the context, described in section 7.2 and set out in table 7.1. 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.

## 11.8 Reviewing wiring, after quick settings

After quick wiring settings are complete, options allow the wiring to be reset, reviewed and amended from a wiring graphic or wiring matrix (enter **return** to skip). These options are the same as described in sections 12.7 and 17.

---

# Chapter 12

## Setting special wiring

Selecting special wiring (enter **s** in section 11.1) allows a greater range of methods for wiring up the network (the wiring scheme) than quick settings in chapter 11. Special wiring is necessary to set random wiring for 2d or 3d networks. The network can be assigned local 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 square+s), 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

Enter **h** for a hypercube in section 12.1. 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. The following values of  $n$  and  $k$  are valid for the hypercube option to appear,

$n$	8	16	32	64	128	256	... and so on
$k$	3 or 4	4 or 5	5 or 6	6 or 7	7 or 8	8 or 9	... and so on

For example, a simple cube,  $n=8$ ,  $k$  must equal 3 or 4. Figures 12.1 and 12.2 give examples in various network-graph layouts (section 20.4).

### 12.2.1 degrading the hypercube

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 a hypercube where  $n=8$  and  $k=4$ , 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%
nhoud mix=44124322 (values shown are examples)
reset-r save-s cont-ret:
```

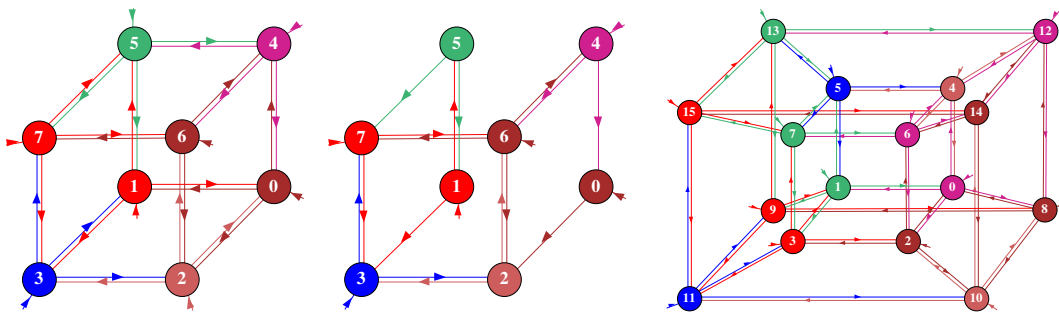


Figure 12.1: Hypercube wiring shown as a network-graph (section 20.4). *Left*:  $n=8$   $k=4$  hypercube, which includes self-links. *Center*: a degraded version of the  $n=8$  hypercube with some links missing. *Right*:  $n=16$   $k=5$  hypercube. The *3d-3* network-graph option was used for the figures, and the graph was rearranged by dragging nodes (section 20.5). The figures should be viewed as if looking up from below. Output directed links are the same color as the source node. To regenerate these layouts automatically, enter *file-f* in section 20.3.2 for the filing prompt (section 35.3) and load the \*.grh layout file, hyp8 or hyp16 (.grh is added automatically).



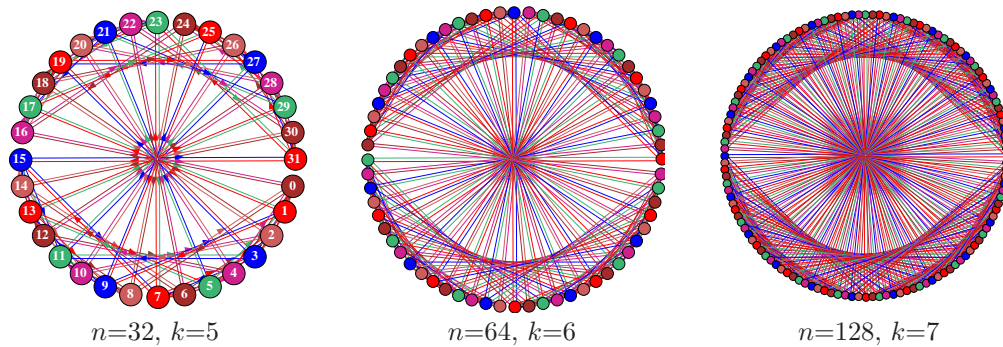


Figure 12.2: Hypercubes for  $n \geq 32$  are difficult to visualise in 3d. These examples show hypercubes as a network-graph with circle layout (section 20.4).

This indicates the actual fraction of remaining wires and the number of and percentage of bi-directional links. 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 9.11.

## 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 local or random wiring, or “hand wiring” in a wiring matrix “spread sheet”. A prompt as follows will be presented,

**hand wire-h**, (*appears in each case below*)

**local 1d-1, random-def:** (*if 1d was set in 12.1*)

or

**local 2d-2 (hex+x square+s), 1d-1, random-def:** (*if 2d was set in 12.1*)

or

**local 3d-3, 2d-2 (hex+x square+s), 1d-1, random-def:** (*if 3d was set in 12.1*)

*options ... what they mean*

**hand wire-h** ... for wiring by hand (section 12.6).

**local-3,2,1** ... for local CA wiring (section 12.4). The wiring would usually be set to match the network geometry set in section 12.1, but its also possible to assign local 1d wiring to a 2d network, and local 1d or 2d wiring to a 3d network.

**return** ... for random wiring with geometry set in section 12.1 — if this is 2d or 3d the size will be set (section 12.3.1). Further refinements are made in section 12.5).

### 12.3.1 Random 2d and 3d

For 2d or 3d network geometry set in section 12.1, if **random** is 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 11.6, 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 the main sequence prompts in chapters 8 and 9.

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**, **2s** or **3** in section 12.3 for local 1d, 2d or 3d (CA) wiring. **2x** or **2s** to forces hexagonal or square wiring and overrides the default, which depends on  $k$  (section 10.1.3). Note that for local 1d wiring in a 1d network, the  $n, k$  parameters set previously in chapters 8 and 9 remain valid. For local 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 local 1d wiring was selected in section 12.3, a subsequent option allows the local wiring to be treated as if it were “random” or nonlocal (always the case for 2d and 3d networks), allowing the local wiring to be altered in various ways, and also for wire moves to be allowed in “learning” (chapter 34). However, the “compression” of attractor basins (section 26.2) may be applied as long as the network retains local 1d wiring. The following prompt is presented,

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

### 12.4.2 Local 2d and 3d

If 2d or 3d local (CA) wiring was selected in section 12.3, further prompts are presented to set the size and neighborhood as for random wiring in section 12.3.1. In fact local 2d and 3d wiring is treated as if it were “random” or nonlocal, allowing it to be altered in any way. This is not the case by default for local 1d wiring (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 center 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-center window **accept defaults-d** is a reminder that at any time further prompts in this sequence can be skipped by entering **d**. Enter **q** 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 predefined 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 17.9.5).

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, then 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 12.3, 12.4 and 12.4. which illustrate how network wiring is represented, explained more fully in chapter 17.

### 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 subsequent changes to be made to the CA wiring, for example, releasing some wires described below.

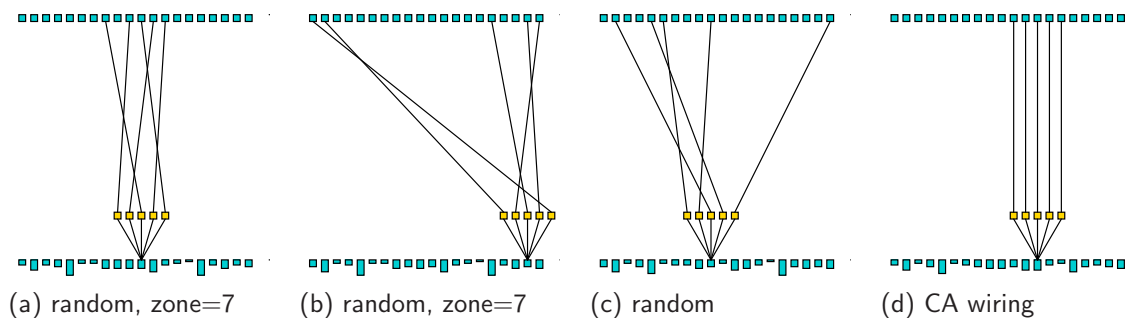


Figure 12.3: Confining 1d  $k=5$  random wiring within a local zone of a set diameter.  $n=15$ . (a) and (b) show random wiring within a 7 cell local zone, where (b) illustrates periodic boundary conditions. (c) shows fully random wiring, and (d) local, CA type, wiring for comparison. Wiring from the pseudo-neighborhood is shown connected to to the previous time-step (see chapter 17)

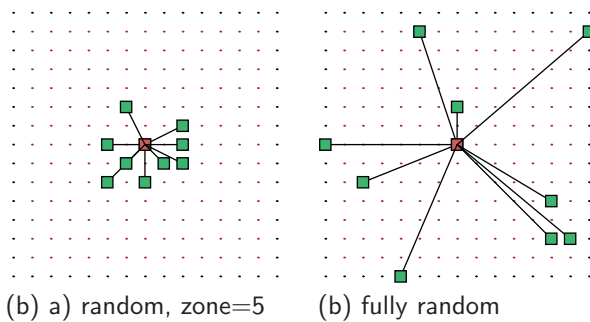


Figure 12.4: Confining 2d random wiring within a 5 cell local zone.  $n=15 \times 15$ . (a) random wiring confined within a 5 cell zone. (c) fully random wiring.

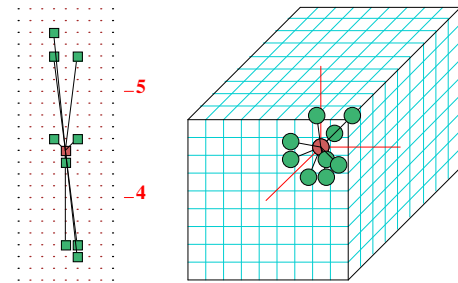


Figure 12.5: Confining 3d  $k=9$  random wiring within 3 cell local zone.  $n=9 \times 9 \times 9$ . *Left*: 2d showing successive layers. *Right*: 3d isometric.

### 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 center 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 (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 horizontal 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. This can be restricted to just part of the network from the wiring graphic (chapter 17).

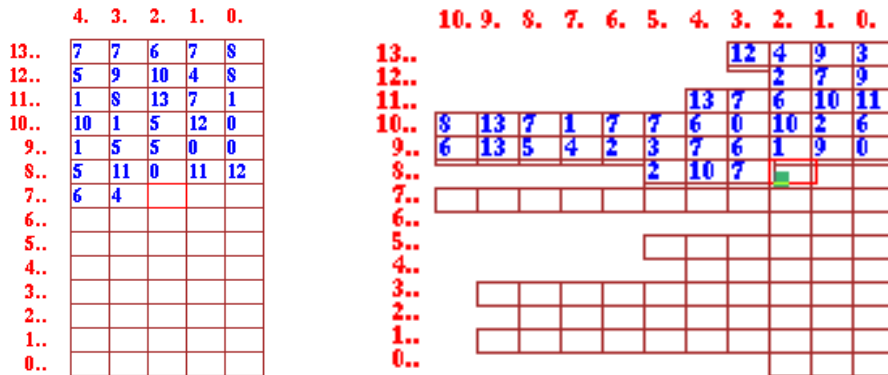
## 12.6 Wiring by hand

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,  $K$  ( $k-1, \dots, 0$ ), Rows give the cell network position,  $N$  ( $n-1, \dots, 0$ ). 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  $x$ , ( $n-1, \dots, 0$ ), 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. Sections 10.2.2 and 10.2.3 explain how to convert between 1d and 2d or 3d coordinates.

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,



(a)  $k=5, n=14$

(b) mixed- $k=3$  to 11,  $n=14$

Figure 12.6: Setting wiring by hand on the blank wiring matrix, shown partly filled. Columns are indexed  $k-1, \dots, 0$ , rows  $n-1, \dots, 0$ . (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,  $n=14$ .

**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 toggle 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 examples*)

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 top or bottom row in the current window, with the arrow keys or **return**, also brings up the preceding or next window. Enter **j** to jump to a new cell index, the following prompt is presented,

**jump to index (1599-0):** (*for a 2d network, 40x40*)

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

Enter **q** to accept the wiring (or **return** on the very last (bottom-right) entry) to conclude matrix entries, and go to the prompt in section 12.7 below. Any blank entries are assigned to position zero.

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 amended 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 (9x9x9), 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 initially, then tailor it with these options. The options also appear after *quick* wiring in chapter 11, and after the rules have been set (chapters 13 to 16).

# Chapter 13

## Rule types

Rules in DDLab can be set according to a number of different types (possibly overlapping) — full rule-table (rcode), k-totalistic<sup>1</sup> (kcode), 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 rulemix 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 rulemix 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 rule-tables (SEED-mode or FIELD-mode) which may be rcode, kcode or tcode, and just totalistic rules (TFO-mode) based on only the shorter kcode and tcode rule-table.

#### 13.1.1 Select full rule-tables, rcode

In SEED-mode or FIELD-modes, all rule types rely on full rule-tables — rcode. Totalistic rules (kcode and tcode) and reaction-diffusion rules will be automatically transformed into rcode. The following prompt is presented,

**totalistic: tcode-t kcode-k (def-rcode):**

*options ... what they mean*

**def-rcode** ... enter **return** for rcode, which allows the rcode method of reaction-diffusion, section 13.8.2.

**kcode-k** ... for tcode, a k-totalistic rule.

**tcode-t** ... for tcode, a t-totalistic rule.

---

<sup>1</sup>Suggested by Antonio Lafusa [5].



### 13.1.2 Select kcode or tcode in TFO-mode

In TFO-mode for kcode or tcode rule-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 11.7)*

*options ... what they mean*

**def-kcode** ... enter **return** for kcode, a k-totalistic rule.

**tcode-t** ... for tcode, a t-totalistic rule.

**outertot-o** ... for *outer*-kcode (section 14.2 which allows the outer-kcode method of reaction-diffusion (sections 13.8.1, 14.2.1)).

**-to** ... enter **t** followed by **o** for an *outer* tcode (section 14.2).

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 conventions 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 generalized 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 rule-tables (lookup-tables) as follows:

- **full rule-tables:** (*not in TFO-mode*) referred to as “rcode” listing the outputs of all possible neighborhood patterns (not in TFO-mode)
- **k-totalistic rules:** referred to as “kcode” — rule-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 rcode.
- **t-totalistic rules:** referred to as “tcode” — rule-tables listing the outputs of all possible totals, where the values in the neighborhood are simply added. tcode can be selected in TFO-mode, or in SEED-mode or FIELD-mode where the tcode is also expressed as rcode. For  $v=2$ , kcode and tcode are identical.

In addition DDLab offers combinations of rules as follows::

- **rulemix:** 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 rulemix.
- **outer-totalistic rules:** (*TFO-mode only, and not for mixed-k*) a number,  $v$ , 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 (section 13.7). For  $v=2$  the game-of-Life can be set in this way, though it can also be set as rcode in section 16.10.

- **reaction-diffusion rules:** or excitable-media [14]. Cells may be either resting, excited, or refractory, and change according to thresholds and values (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 rcode (section 13.8.2) or from an outer-kcode (section 14.2.1). Note that outer-kcode allows a greater range of  $[v, k]$  than a rcode, but the latter allows a rulemix that can include more than one reaction-diffusion rule in the network.

### 13.2.1 Rule-table size implications

The lengths of rule-tables,  $S$ , depend on  $v$  and  $k$ , where rcode > kcode > tcode,

$$\begin{aligned} \text{rcode} \dots S &= v^k \\ \text{kcode} \dots S &= (v+k-1)! / (k! \times (v-1)!) \\ \text{tcode} \dots S &= k(v-1) + 1 \end{aligned}$$

This has implications on the  $k_{Lim}$  (section 7.2). In SEED-mode or FIELD-mode, kcodes and tcodes will be implemented as the equivalent full rcode (the longest rule-table), 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 rule-table — rcode

A full rule-table (rcode) for binary systems ( $v=2$ ) has  $S = 2^k$  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 Boolean value order<sup>2</sup> following Wolfram [28], 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 referred to by their hex rule numbers,  $k \leq 3$  rules are usually referred to by their more familiar decimal rule numbers, especially binary  $k=3$  so called “Elementary Cellular Automata” — for example ECA 60 is shown 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, 60 in decimal

<sup>2</sup>The rcode rule-table order can be inverted — section 18.3

In DDLab the neighborhood configurations are rotated and displayed vertically 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
      :
      :
k index 2 - 11110000
        1 - 11001100
        0 - 10101010
        -----
      11000001 - rcode, decimal 193, or hex c1

      31..... ..... 0 - rcode index
      :
      :
k index 4 - 11111111 11111111 00000000 00000000
        3 - 11111111 00000000 11111111 00000000
        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 (section 14.11).

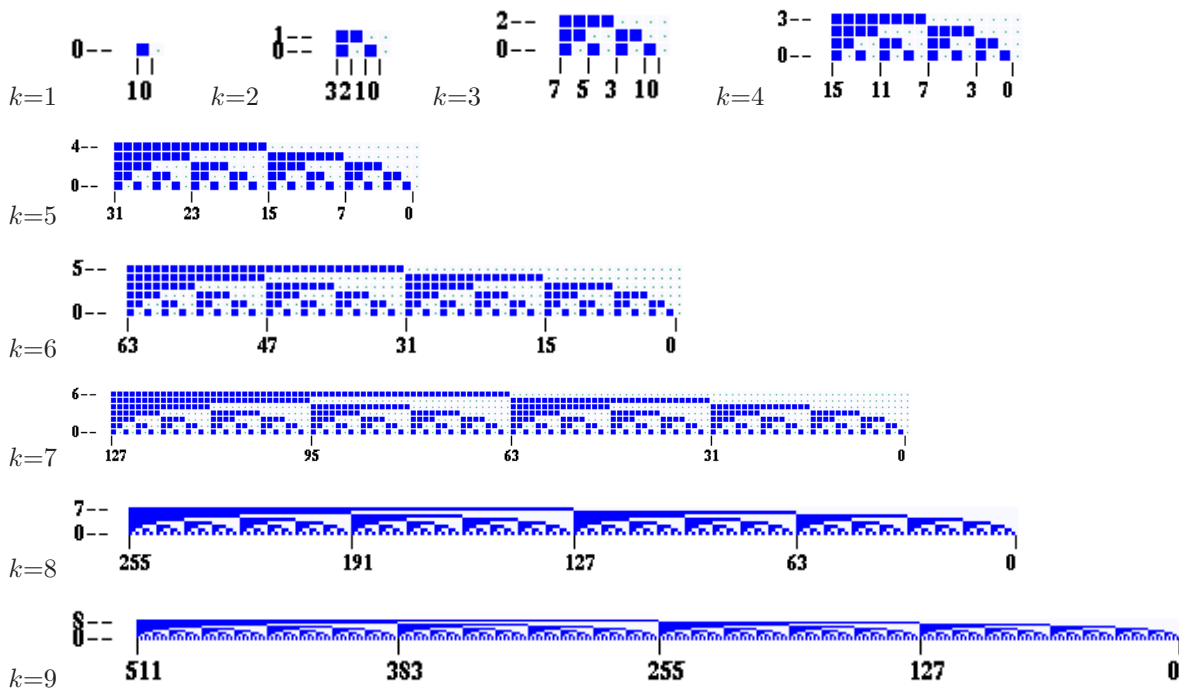


Figure 13.1: The binary neighborhood matrix,  $k=1$  to 9, as displayed in DDLab.

## 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 (for  $v \geq 3$  they are different — section 13.6).

A tcode rule-table (tcode-table) has  $k+1$  bits representing the possible totals, set out in reverse value order, from  $k$  to 0, where 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 rcode (section 13.3). Here is an example for  $k=5$ ,

```

5 4 3 2 1 0 - totals
- - - - -
1 1 0 0 1 0 - output = tcode = kcode, 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  $k5$  tcode-table 111000 is the majority rule.

## 13.5 Multi-value full rule-table — rcode

A full rule-table, rcode, for multi-value systems  $>2$  ( $v=2$  to 8 in DDLab), has  $v^k$  entries. Table 13.1 shows how rcode increases exponentially with  $v$  and  $k$ , and also the “safe” limits of  $k$ . Larger but risky  $k_{Lim}$  (table 7.1) are possible if “exLimits” was activated in section 7.1.1.

$k$	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	15625			
$v=6$	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 rule-tables for  $v=3$  to 8, against  $k$ , showing “safe” values of  $k$ .

The convention in DDLab is to list neighborhood configurations from left to right in reverse  $n$ -ary value order<sup>3</sup> so the all-max neighborhood is on the left. Each neighborhood is assigned an output  $[0,1,\dots,v-1]$ , and the resulting  $n$ -string defines one of the  $v^{v^k}$  possible rcodes. The rcodes may also be expressed in decimal (if applicable — section 16.6) or in hexadecimal. An example for a  $v3k3$  rcode 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

```

<sup>3</sup>The multi-value rcode rule-table can be inverted – section 18.3

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 vertically for compactness making a so called “neighborhood matrix”, showing also the  $k$ -index. The  $n$ -ary string can be assigned in the corresponding order to make the rcode.

```

    26.....0 - rcode index
      |
      | 2 - 22222222211111111000000000
k index 1 - 222111000222111000222111000
      0 - 210210210210210210210210210
          |
          | 002121012212110211000202201 - output string = rcode, hex 026469949408a1

```

### 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 (section 14.11).

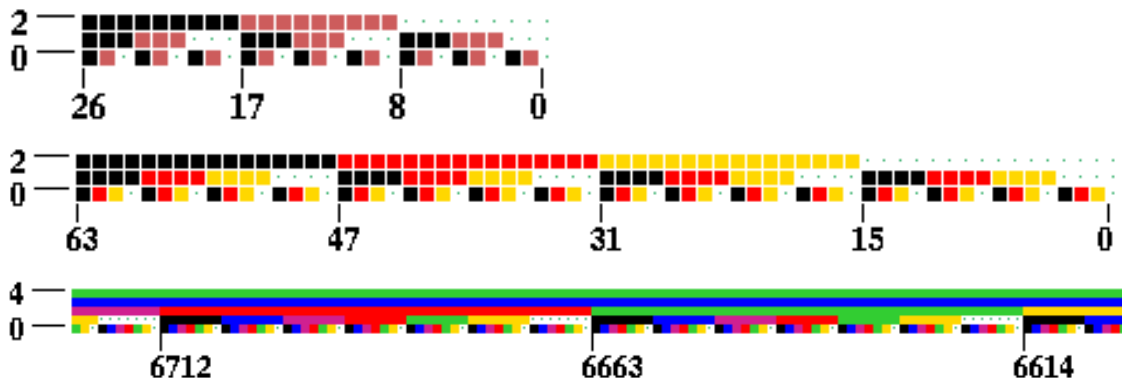


Figure 13.2: Examples of the multi-value neighborhood matrix, colored according to the value color key (section 7.1). *Top*:  $v3k3$ , *Center*:  $v4k3$ , *Bottom*:  $v5k7$  central part only.

## 13.6 Multi-value totalistic rules, tcode and kcode

For multi-value systems, if  $v \geq 3$ , totalistic rules separate into two types — k-totalistic rules (kcode), and t-totalistic rules (tcode). For  $v=2$ , kcode and tcode are identical.

### 13.6.1 k-totalistic rules - kcode

The kcode rule-table (kcode-table) is a list of the outputs for all possible combinations of value-frequencies in the neighborhood. Each combinations is represented by a string of length  $v$  (shown below vertically from  $v-1$  down, for  $v=3$ ,) 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.

```

                27.....0 <--kcode index
                |
v=3 values > 2: 655444333322221111110000000 < frequency strings      6    0
            > 1: 0102103210432105432106543210 < of 2s, 1s, 0s,      from 0 to 0
            > 0: 0010120123012340123450123456 < shown vertically    0    6
                |||
                0022000220022001122200021210 <--kcode, outputs [0,1,2]
    
```

The ordering of the pattern strings themselves depend on their  $v$ -ary value, with the higher value on the left. This can be effectively inverted to allow kcode expressed in the opposite convention to run as intended (section 31.2.10).

In the example above for the  $v3k6$  “Beehive rule” [44] (also shown as a matrix 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 rule-table, rcode (section 13.5).

Kcode rules are isotropic because they depend only on the frequency of each value (or color) in the neighborhood — the positions of values are irrelevant, so that symmetric space-time pattern must conserve their symmetry.

The size of a kcode-table,  $S = (v + k - 1)! / (k! \times (v - 1)!)$ , is much shorter than a rcode-table. The values of  $S$  for different  $v$  and “safe”  $k_{Lim}$  in TFO-mode are shown in table 13.3 — table 13.4 shows all “safe” alues of  $S$  for increasing  $v$  and  $k$ . Larger but risky  $k_{Lim}$  (table 7.1) are possible if “exLimits” was activated in section 7.1.1.

$v$	$k_{Lim}$	kcode size
2	27	28
3	27	406
4	26	3654
5	25	23751
6	17	26334
7	13	27132
8	11	31824

Table 13.3: Kcode lookup-table size for  $v$  and “safe”  $k_{Lim}$ .

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	$k$	15	16	17	18	19	20	21	22	23	24	25	26	27
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	27	28	
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	378	406	
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	3654		
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  $v=2$  to 8 showing “safe” values of  $k$ .

### 13.6.2 kcode $v=3$ matrix

For ternary ( $v=3$ ) systems, kcode can usefully be show as a 2d matrix [45], 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 table 13.5 for  $k=6$  and  $k=7$ .

					$j$			
		0	1	2	3	4	5	6
	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						

						$j$			
		0	1	2	3	4	5	6	7
	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							

Table 13.5: Matrix representations of  $v=3$  k-totalistic rules. *Left:*  $v3k6$  Beehive rule [44] and *Right:*  $v3k7$  Spiral rule [45, 46], both on a 2d network with hexagonal tiling [44, 45].

### 13.6.3 t-totalistic rules - tcode

Tcode depends 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-table,  $S = k(v - 1) + 1$ , is much shorter than kcode (section 13.6.1), but the tcode  $k_{Lim}$  is nonetheless set at the same level as kcode in TFO-mode (“safe” or “risky” — table 7.1). The values of  $S$  for different  $v$  and safe  $k_{Lim}$  are shown in table 13.6 below,

$v$	$k_{Lim}$	tcode size
2	27	28
3	27	55
4	26	79
5	25	101
6	17	86
7	13	79
8	11	78

Table 13.6: Tcode lookup-table size for  $v$  and “safe”  $k_{Lim}$ .

To construct tcode, each total, set out in reverse value order,  $S-1$  to 0, is assigned an output  $[0,1,\dots,v-1]$ , 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, rcode (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 rulemix (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 utilizes the rulemix functionality, it is not a proper rulemix.

The method works with any  $k$ , but makes most sense if the central cell is not wired to itself — is empty in the neighborhood (section 10).

For the classic binary game-of-Life (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 neighbors       ←  $k=8$  outer-neighborhood  
 for  $v=1$ : 000001100 - survival: 2 or 3 live neighbors

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 rule-table, rcode (section 16.10).

## 13.8 Reaction-Diffusion dynamics

Reaction-Diffusion or excitable media dynamics [14], 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  $\geq 2$ , which cycle between each other<sup>4</sup>.

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,

resting(0): if within ( $t$ )  $\rightarrow$  excited(1)  $\rightarrow$  refractory( $2 \rightarrow 3 \rightarrow \dots \rightarrow v - 1$ )  $\rightarrow$  resting(0)

In DDLab, reaction-diffusion can be set as outer-kcode in TFO-mode, which allows greater  $[v, k]$ , or as a full rule-table — rcode.

The resulting dynamics, in 2d and 3d, 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 ( $\lambda$  parameter) can be set in section 16.3.1. Interesting results (as illustrated in figures 13.3 and 13.4) can be achieved not only for CA, but also for DDN were the random wiring is confined to a tight local zone (section 12.5.2 and figures 12.4, 12.5).

<sup>4</sup>For binary networks ( $v=2$ ) the refractory type would be missing, so this would not be reaction-diffusion in its proper sense, though the option is still present in DDLab.



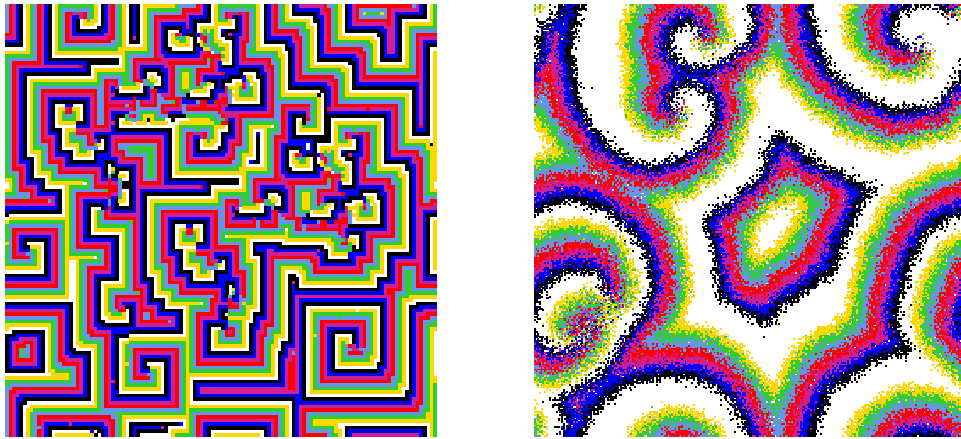


Figure 13.3: Reaction-Diffusion dynamics, 2d. Left: *v8k8*, threshold interval is 1 to 6, 122x122 square lattice. Right: *v8k11*, threshold interval is 2 to 7, 255x255 square lattice, random connections within a 24 diameter local zone.

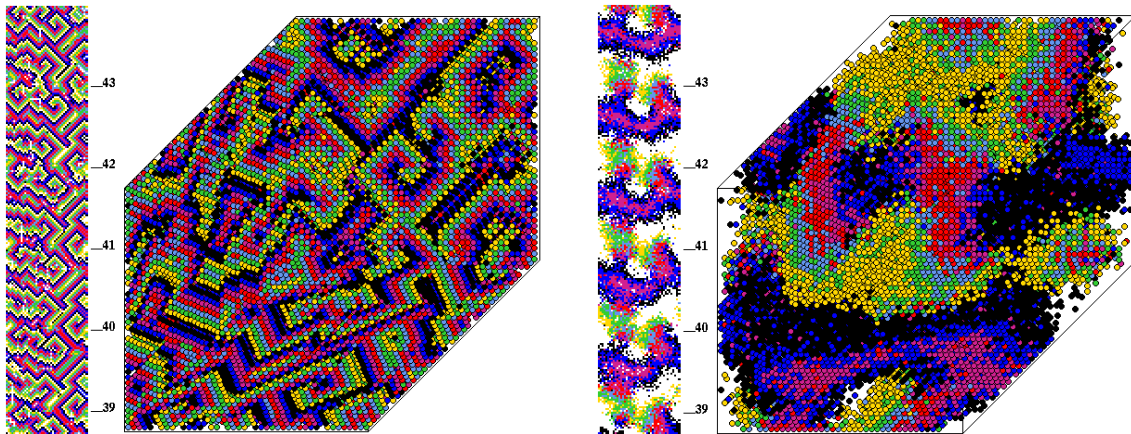


Figure 13.4: Reaction-Diffusion dynamics, evolved 3d snapshots  $44 \times 44 \times 44$ , *v8k11*, threshold interval is 2 to 6, with a seed density of 33%. Each 3d snapshot has its 2d version, the top 5 levels, beside it. Left: a local CA 3d neighbourhood, Right: random connections within a 7 cell diameter local 3d zone.

### 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 (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<sup>5</sup>. The way this works is as follows, where resting=0, excited=1,

<sup>5</sup>As noted in section 13.7, outer-totalistic rules are themselves a quasi-rulemix, using the rulemix functionality.

refractory $\geq 2$ : kcode(0) holds the threshold interval to change the cell from resting to excited (0 $\rightarrow$ 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.

### 13.8.2 Reaction-Diffusion from a full rule-table — rcode

Select rcode which implements any logic, by entering **return** in section 13.1.1. Then enter *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 rulemix.

### 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

## Rulemix 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 rulemix (or rule scheme), which may be a rcode-mix, kcode-mix or tcode-mix. If the network has heterogeneous neighborhoods sizes (a  $k$ -mix, chapter 9 and section 11.7) then there must be a rulemix (by default) — in this case section 14.1 below does not apply. For just one rule enter **return** at the prompt in section 14.1 and continue with chapter 16.

This chapter describes the rulemix 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<sup>1</sup>.

A rulemix can be assigned from ...

- the whole of rule-space (rule-spaces for a  $k$ -mix)
- a limited subset of rules<sup>2</sup> to restrict the range of rules in the rulemix. The number of rules in the subset can be anywhere from 1 to  $n$  (network size), but with an upper limit of 200. Rules from the subset can be assigned to the network at random, or in repetitive blocks — thus implement so called “hybrid CA” or HCA.

Rules in a rulemix or a limited subset can be set automatically at random with a predefined density-bias of non-zero outputs ( $\lambda$ -parameter), and rules can be confined to various sub-categories. Alternatively, rules can be set individually “by hand” allowing almost any combination, and the rulemix can be loaded from a previously saved file.

This chapter also describes altering the presentation of the neighborhood matrix (section 14.11), and listing Post-functions, which are included in the initial rulemix prompt.

---

### 14.1 Single rule or rulemix, and other options

*not for a  $k$ -mix (unless all  $k$ 's equal, section 14.6.7), or outer-totalistic rules in TFO-mode*

After selecting a rule type in section 13.1, the next top-right prompt selects a single rule or a rulemix, 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 ( $\lambda$ -parameter). The prompt also differs somewhat according to the rule type,

---

<sup>1</sup>Reaction-diffusion or Life-like rules can also be set as full rule-tables (rcode), allowing inclusion in a rulemix.

<sup>2</sup>This subset option does not apply to a  $k$ -mix, or to outer-totalistic rules in TFO-mode.

*For rcode, kcode and tcode based on full rule-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 rule-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:

Enter **return** to skip all these rulemix options (and this chapter), and continue for a single rcode, kcode or tcode, in chapter 16. The rulemix options are summarized below.

### 14.1.1 Summary of rulemix and other options

A summary of the options in section 14.1 is as follows,

*options ... what they mean*

**bias-density-s** ... to change the density-bias — the fraction of non-zero values in the rule-table (section 14.1.2).

**load rulemix-l** ... to load rule a scheme (rulemix) — rcode-mix .r\_s, kcode-mix .r\_v, or tcode-mix .r\_t. (section 35.3, and chapter 19).

**list Post-P** ... *(not in TFO-mode)* to list Post functions (section 14.12).

**nhood-matrix-a** ... *(not in TFO-mode)* to amend the presentation of the neighborhood matrix (section 14.11).

**no limit-n** ... to set an rcode-mix, kcode-mix or tcode-mix, taken from the whole of rule-space (section 14.4.1).

**up to 200** ... enter 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). The maximum size of the limited subset of rules (**set limit**) is 200, unless the network size  $n < 200$  — then the maximum **set limit** is  $n$ .

### 14.1.2 Density-bias ( $\lambda$ -parameter)

Enter **s** in section 14.1 to change the density-bias — the fraction of non-zero values in the rule-table (the  $\lambda$ -parameter) but expressed as a percentage. This is applied probabilistically, so each rule (rcode, kcode or tcode) in a rulemix 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 rulemix or subset of rules (sections 14.5, 14.4.2).
- a rulemix 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 section 16.3.1.

## 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-rulemix 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  $v=8$ )*

**OuterKcode(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 rulemix, but although the method utilizes the rulemix functionality, this is not a proper rulemix (section 13.7).


### 14.2.1 Setting reaction-diffusion by outer-kcode

*for reaction-diffusion from rcode 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 (sections 13.8, 13.8.1 and figure 13.3). Then set the threshold interval as described in section 13.8.3. An alternative method for reaction-diffusion is from the full rule-table — rcode, which also allows reaction-diffusion rules to form part of a rulemix, (section 13.8.2).

### 14.2.2 The game-of-Life and other Life-like rules by outer-kcode

*for Life from a full rule-table (rcode) see section 16.10*

If  $k=8$  on a 2d square lattice, giving this neighborhood  — John Conway’s game-of-Life [10] can be set as an outer-kcode. Any other Life-like<sup>3</sup> rule from the “Life family” can also be set, as well as Life-like rules with neighborhoods other than the Moore neighborhood, or with  $v > 2$ . This can also be done from a full rule-table (rcode) 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,

**Life k=8 (def: survival 2,3, birth 3,) accept-ret amend-a:**

<sup>3</sup>[http://en.wikipedia.org/wiki/Life-like\\_cellular\\_automaton](http://en.wikipedia.org/wiki/Life-like_cellular_automaton)

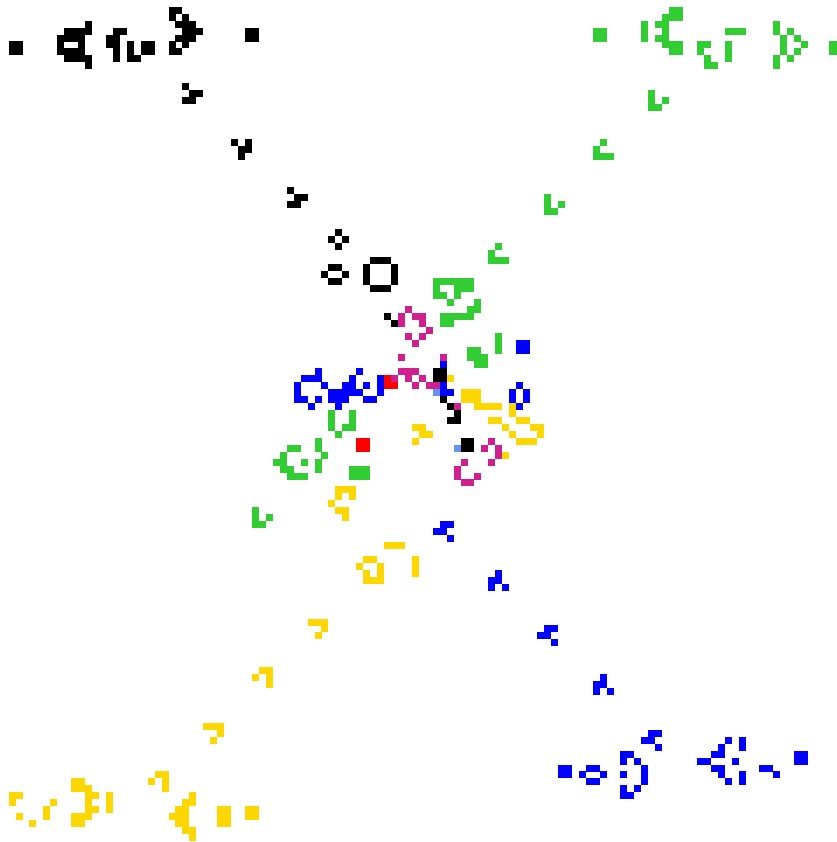


Figure 14.1: The game-of-Life (23/3) applied to a  $v=8$  CA. The algorithm in DDLab generates 8 outer kcodes giving similar dynamics to classical binary Life, but including 7 colors + background. In this example 4 glider guns are located in each corner on a  $122 \times 122$  lattice, shooting different colored gliders towards the center. The seed file is `Lguns_v8.eed`.

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 rulemix by hand. For  $v > 2$ , for a (23/3) Life setting, the algorithm in DDLab generates an equivalent rule-table giving the same dynamics as classical binary Life, but including  $v-1$  colors plus the background, as in figure 14.1.

As well as using the prompts 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 11.7, 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 effectively  $k=0$  (section 9.2). 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** to accept. Note that if the effective wiring to a cell is  $k=0$ , it will appear disconnected in the 1d wiring graphic (section 17.6), as illustrated in figure 14.2.



Figure 14.2: Effective neighborhood  $k=0$ , as shown in the 1d wiring graphic (section 17.6) with no connections to the previous time-step.

### 14.4 Methods for setting the rulemix or rule-subset

A rulemix, rule-subset (or quasi-rulemix) will be set if,

- **n** for *no limit* was selected in section 14.1 — this sets a rulemix
- a number, 1 to  $n$  (upper limit 200) for a rule-subset was selected in section 14.1 — this sets the subset size.
- the network has mixed- $k$  (set in chapter 9 or section 11.7) — this forces a rulemix.
- outer-totalistic was selected in section 13.1.2, followed by the outer-totalistic reminder in section 14.2 — this sets a quasi-rulemix of  $v$  rules.

There are two strategies for creating a simple rulemix.

- The direct strategy, the only possibility for a  $k$ -mix, otherwise 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  $s$ : 1 to  $n$ , upper limit 200). This first creates a limited subset of rules from which rules will be assigned at random to the network, or sequentially in repeating blocks of size  $s$ , so the final rulemix can be restricted to just a few rules - or even just one rule (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,

**RcodeMix: 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 rule-tables (rcode) 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  $s$  of the subset of rules. Rules from this subset will be automatically assigned at random to the network, or sequentially in repeating blocks of size  $s$ . The prompts are similar to section 14.4.1 but without **maj** and **canalyzing**, and **Alt** for tcode.

For rcode the prompt is,

**RcodeSubset(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 rule-tables (rcode) in section 13.1.1, the prompts are,

**KcodeSubset(5): select by hand-h, Alt-A rnd-(def):** *(for kcode, for subset size 5)*  
or  
**TcodeSubset(33): select by hand-h rnd-(def):** *(for tcode, for subset size 33)*



### 14.4.2.1 Setting a rulemix indirectly - apply the rule-subset

Once the rule-subset has been selected, a further prompt is presented,

**2 Rcodes were set** (for a subset of 2 rules, or Kcodes, Tcodes)  
**apply as repeating blocks-b, or randomly-(def):**

Enter **return** to set rules from the subset at random. Enter **b** to set rules sequentially in repeating blocks equal to the subset size  $s$ . In the example above, the two rules in the subset will be set alternately in the network, which can sometimes lead to interesting results for so called “hybrid CA” or HCA.

### 14.4.3 A rulemix 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 rulemix (as in figure 14.3), select a rulemix with a limited subset of rules consisting of just one rule, i.e. enter **1** at the prompt in section 14.1.1, and set the rule *by hand* (section 14.6). This fools DDLab into assuming the network has a “rulemix” when in fact all the rules are the same.

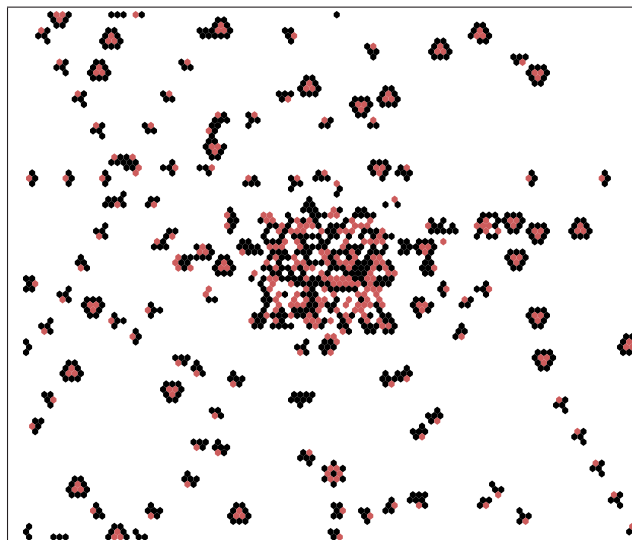


Figure 14.3: A 2d CA set up with the spiral rule [45]  $100 \times 100$  perturbed by a central  $20 \times 20$  homogeneous block of chain rules that have been inserted. To do this first set up as a rulemix consisting of just one rule, then create a rule block (sections 17.7.5 and 17.9.10).

---

## 14.5 Rulemix - 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.

- options ... what they mean*
- rnd-(def)** ... enter **return** to assign rules entirely at random from the whole of rule-space, for rcode, kcode or tcode. The density-bias ( $\lambda$ -parameter) can be preset in section 14.1.2
- maj-m/u** ... enter **m** to assign majority (voting) rules (section 14.7). Enter **u** (*not for tcode*) to assign majority rules but with the outputs from uniform neighborhoods flipped or shifted (section 14.8).
- Alt-A** ... (*not for tcode*) to assign rcode or kcodes at random from the subset of “Altenberg” rules, (section 16.9)  
*the options below for rcode only*
- chain-c** ... to assign rcode at random from the subset of “chain” rules, (section 16.11).
- iso-i** ... to assign isotropic rcode at random, where rotated and reflected neighborhoods (in 1d, 2d and 3d) have the same output.
- Post-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.
- canalyzing-C/+C** ... enter **C** to see and amend rcode canalyzing inputs, or enter another subcategory followed by **C**: **mC**, **uC**, **AC**, **cC**, **iC**, or **PC**. Canalyzing inputs are described in chapter 15.

## 14.6 Rulemix by hand

A succession of rules can be set individually, by hand, to create a rulemix, rule-subset, or outer-totalistic rules. 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 context dependent reminder appears, for example,

*for a rule subset*

**by hand: subset of 22 rcodes** (*or kcode, tcode*)

*for a rulemix of the whole network*

**by hand: kcodemix of 150 kcodes** (*or rcodes, tcodes*)

*for outer-totalistic rules ( $v=2$ ) — TFO-mode only*

**by hand: outertot of 2 kcodes** (*or tcodes*)

*for a kmix (the whole network) — 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 summarized in section 16.2.

## 14.6.3 By hand options

The rule at each “index” is set in turn, starting from index 0, where “index” relates to the context selected — either the whole network, or the rule subset, or outer totalistic. A top-right prompt keeps track of the index, the range of  $k$  for a  $k$ -mix, the current value of  $k$ , the rule-type, and the current *by hand* selection method.

For a  $k$ -mix, if a rule has already been set for the current  $k$ , the prompt start as follows,

**index=7 k(3-9) (v3k4 rcodemix), same as last-s, all remaining-a:** *(for example)*

Options **s** and **a** to copy previous rules are described in (section 14.6.6). Without a previous  $k$  rule there are no copy options — the  $k$ -mix prompt is as follows, for example,

**index=7 k(3-9) (v3k4 rcodemix)** *(or kcodemix, tcodemix)*  
**hand=r, change-(any valid entry) or index-q:** *(if r is the current selection method)*

For a homogeneous- $k$  network, the prompt is as follows, for example,

**index=12 (v3k4 rcodemix)** *(rnd all-a if hand=r, m, A, c, i only)*  
**hand=A, change-(any valid entry) or index-q, rnd all-a:**

Opinion **a** is described in (section 14.6.5). Enter **return** to set the rule with the current selection method, possibly with intermediate steps. The methods are similar to setting a single rule (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 **q** to change the current selection method or jump to a new index (section 14.6.4).

## 14.6.4 Change the selection method or rule index

To change the selection method, enter any valid option in sections 14.6.2 or 14.6.3.

If **q** is entered in prompt 14.6.3 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 entered above, the following top-right prompt appears, for example,

**this index=16, enter new index 0-16:**

Enter a number to change the rule index.

If *selection-s* is entered above, the following top-right prompt appears, where the lower line gives a list of valid selection methods. The options included in the list depend on the rule type and *k*, and correspond to the main sequence single rule options (sections 14.6.2 and 16.1), for example,

*for rcode*

**index=16 (v5k4), change selection method (now r), abort-q**  
**as main sequence choices - e/f/x/m/u/A/L/c/r/i/R/b/h/d/p/l/v:**

*the second line for kcode*

**as main sequence choices - e/f/x/m/u/A/r/b/h/d/p/l/v:**

*the second line for tcode*

**as main sequence choices - e/f/x/m/r/b/h/d/p/l/v:**

Enter any one of the listed choices to change the selection method which becomes the new default. The selection methods are described in detail in chapter 16 and summarized in section 16.2.

### 14.6.5 Complete the rulemix automatically

For a rulemix with homogeneous-*k*, if the current selection method is setting rules at random, or at random with constraints (section 16.2), the following option is included in the prompt in section 14.6.3,

... **rnd all-a:** (*if hand= 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 according to the density-bias ( $\lambda$ -parameter) if **hand=r** (section 16.3.1).

### 14.6.6 Copy rules automatically for a *k*-mix

While setting rules by hand for a *k*-mix, a rule set previously with a given *k* can be copied. The following option is included in the prompt in section 14.6.3,

... **same as last-s, all remaining-a:** (*if a rule for the current k was set previously*)

Enter **s** to copy to the current cell index, or **a** to also copy to all the remaining cells in the network with the same *k* — these cell indexes will be skipped in the remaining rule assignment procedure. Enter **return** to ignore these options 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 9.3 ...

**set k-mix: load-l hand-h specify-s rnd-(def):**

... select **s** to set the percentage of *k*'s required (section 9.7.1), 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 9.10) for example,

```
... set greater max k-max (def 3, limit 9):7
```

After further options, DDLab will recognize if all  $k$ 's are the same, and present the single rule or rulemix prompt (section 14.1) — for a single homogeneous rule enter **1**.

## 14.7 Rulemix - 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 Rulemix - 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 16.9).

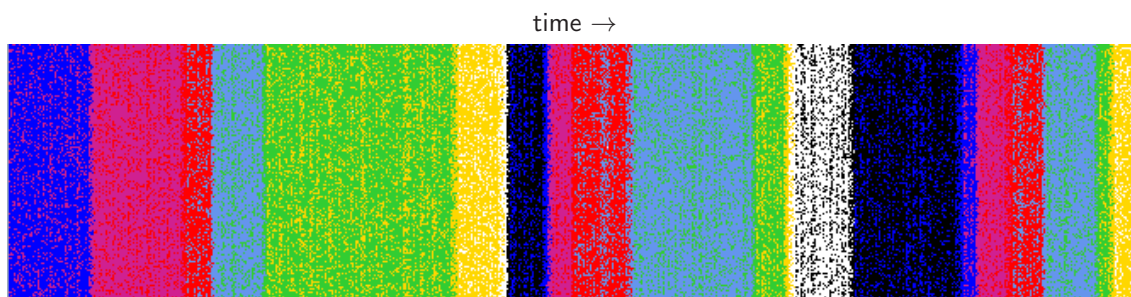


Figure 14.4: Shifted majority kcode-mix with random wiring,  $v8k11$ , where uniform neighborhoods have their outputs shifted to a different color. The 1d space-time pattern ( $n=150$ ) shows zones of color density that remain stable for an unpredictable number of time-steps before flipping to a different quasi-stable regime. The image shown is rotated by  $90^\circ$  so the time flows from left to right.

---

## 14.9 Rulemix for large networks, or large $k$

Assigning the rulemix 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 rulemix 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 rulemix as is. For kcode or tcode, enter **0** to force all 0s->0, or **return** to leave rulemix 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 rule-tables (rcode) only, not in TFO-mode*

If **a** to amend the neighborhood matrix ( $k$ -matrix) presentation (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)
```

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 neighborhood 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.



Figure 14.5: Part only of the multi-value neighborhood matrix, colored according to the value color key (section 7.1), this example for  $v8k4$ , set at 33%, and scale=12.

## 14.12 List Post functions

for rcde only, not in TFO-mode

Post functions [26] 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 generalized 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  $[v-1, v-2, \dots, 0]$  or just  $[1, 0]$  in the Boolean case, and the Post class is  $[2, 3, 4, \dots, i]$ , where  $i$  stands for *all* or “infinity”, a rule-table (function) belongs (non-exclusively) to  $A[u]2, A[u]3, \dots, A[u]i$ , if any  $[2, 3, \dots, all]$  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

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,

```

      _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 tcode-sample=x \_or (full list)

```

Table 14.1: Post function list and data output for tcode-space  $v2k3$ , no restriction, with explanatory notes. Only rule-tables  $\leq 128$  are displayed. The last line is the data summary.

```

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: 01110010111110001000000100000000 A[1]2 C=0 P=0.625 Z=0.5625
38222: 11011010000000001101110000000000 A[1]i C=1 P=0.65625 Z=0.570312
38161: 01011011100011010001000000000000 A[1]2 C=0 P=0.65625 Z=0.617188
28130: 00000100100000001100000011000100 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: 11000100001001110000000000000000 A[1]i C=1 P=0.78125 Z=0.4375
11724: 00101010000000101000000010001000 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  $v2k5$  rcde-space, restricted to Post value 1, the sample index is on the left. For explanatory notes see table 14.1.

```

1553: 221222121222212121102 A[0]3 C=0 Z=0.26749
1418: 200000000000200100220 A[1]i C=0 Z=0.152263
1145: 200000002022022000020 A[1]i C=0 Z=0.251029
924: 122112122212101111000 A[0]2 C=0 Z=0.419753
656: 022000022000022100020 A[1]i C=0 Z=0.27572
473: 111122221221211222021 A[0]2 C=0 Z=0.358025
422: 121210101101010101111 A[2]2 C=0 Z=0.366255
414: 020200000212022002000 A[1]3 C=0 Z=0.283951
222: 010212000111001000101 A[2]2 C=0 Z=0.366255
202: 222222020221022112022 A[1]2 C=0 Z=0.407407
197: 022022202201202202200 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.

**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 >  $2^{32}$ )

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 [ $v - 1, v - 2, \dots, 0$ ]. 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 filename **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.



## Chapter 15

# Setting Canalization in a random rcode-mix

*not in TFO-mode.*

The dynamics on a random rcode-mix can be biased towards order by including varying proportions of “canalizing” inputs, with applications in modelling genetic regulatory networks [16, 36].

A canalizing input may be defined as follows: if a particular value  $(0, 1, \dots, v-1)$  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, but 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) decreases exponentially (section 24.9.5), so the probability of setting such a rule at random decreases accordingly.

Traditionally, canalization applies to Boolean rules, but the same definition has been generalized 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

If not in TFO-mode, canalizing can be selected at various points in DDLab.

#### 15.1.1 Selecting canalizing from the rcode-mix

When setting the rcode-mix directly in section 14.4.1, at the top-right prompt,

```
RcodeMix: select by hand-h, maj-m Alt-A chain-c iso-i Post-P  
canalizing-C/+C, rnd(def): (canalizing-C/+C not for kcode or tcode)
```

Enter **C** to apply canalizing to random rules, or **mC**, **AC**, **cC**, **iC** (but not **PC**), to apply canalizing to biased random rules. The (biased) rules will be set first, then modified by cumulative random mutation to achieve the required canalization.

### 15.1.2 Selecting canalizing from wiring graphic — transform rule

Once the rulemix has been set, canalizing can be reached from the wiring graphic options (section 17.4). At the wiring graphic option,

... rule: **Save/rev/trans-S/v/t** ...

... enter **t** for the *transform rule* options (section 18.1). 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 18.6. The *transform rule* prompt also appears automatically after a single rcode is set in the main prompt sequence, allowing canalizing to be set for that rule.

### 15.1.3 Selecting canalizing from the Derrida plot

Canalizing can be selected from the Derrida plot (chapter 22). The following top-right prompt appears once the Derrida plot is complete (section 22.5),

**Derrida plot complete  
reset-r canalizing-C:**

Enter **C** to reset the canalizing for a new Derrida plot.

## 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, the fraction of network elements with at least one canalizing input. Enter **return** (the default) to specify the fraction of 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): (or c-genes)  
or  
mixed k, c-inputs: redo-q number-n %-(def): (or c-genes, for mixed-k networks)**

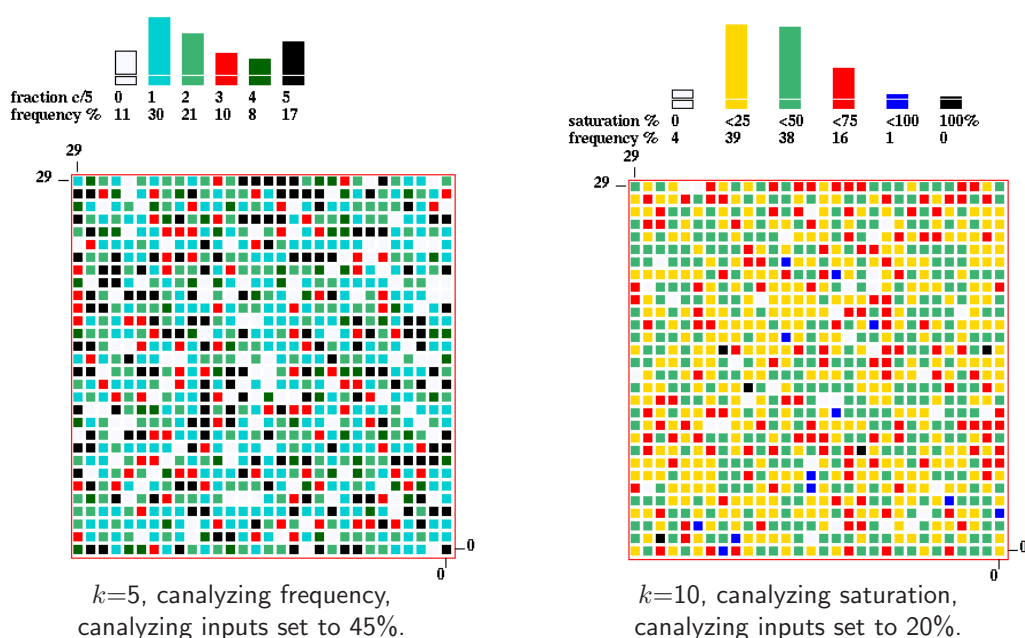


Figure 15.1: Analyzing frequency/saturation for homogeneous- $k$  networks,  $n=30 \times 30$ , showing the degree of canalyzation of each gene according to a color code, and a histogram of canalyzation in the network — the colors in the network and histogram correspond. *Above Left*: canalyzing frequency histogram. *Above Right*: canalyzing saturation histogram (because  $k \geq 10$ ). The colors in the network and histogram correspond.

Enter **return** (the default) to set a percentage of the total genes/inputs as canalyzing, 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 Canalyzing - homogeneous- $k$

The next prompt gives the fraction and percentage of the current canalyzing genes and inputs, and asks for the required canalyzing settings, for example,

```
c-inputs=0/4500=0.0% genes=0/900=0.0% (for a 2d network,  $k=5$ ,  $30 \times 30$ )
set new % c-inputs, redo-q accept-ret: (values depend on the earlier choices)
```

If, say, **45** (for 45%) is entered. The prompt will reappear with updated information,

```
c-inputs=2025/4500=45.0% genes=793/900=88.1%
set new % canalyzing inputs, redo-q accept-ret:
```

Enter a new canalyzing setting, or **return** to accept — the program will continue with options to review network architecture described in chapter 17, or enter **q** to revert to the first canalyzing prompt (section 15.2).

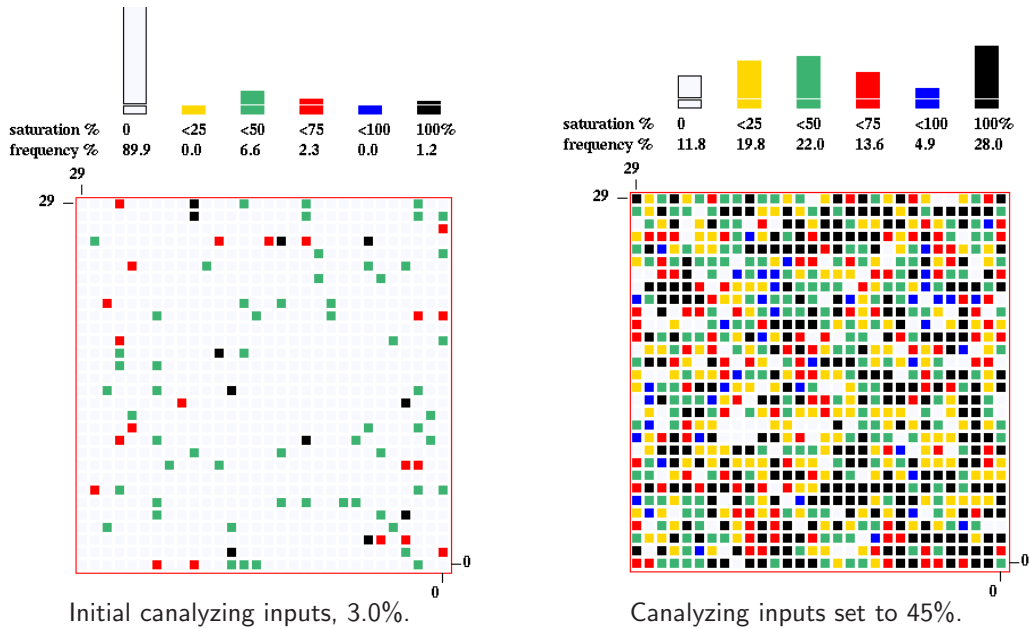


Figure 15.2: Canalyzing saturation for two networks with the same (roughly even)  $k$ -mix 3-7, with the canalyzing saturation histograms above, relating to section 15.5.  $n=30 \times 30$ . *Left*: The network showing the initial, expected, canalyzing. *Right*: The canalyzing inputs set to 45%. The colors in the network and histogram correspond.

At the same time as the prompt appears, a window in the lower right hand corner of the screen gives a 2d graphical display of the degree of canalyzation of each gene, irrespective of the network's native dimensions (figure 15.1).

Above this is a histogram of canalyzation in the network. For  $k \leq 9$  the “canalyzing frequency” histogram shows the frequency of each degree of canalyzation, 0 to  $k$  (figure 15.1 *Above Left*). For  $k \geq 10$  or for mixed- $k$  (section 15.5) a different histogram, “canalyzing saturation” shows the frequency of different percentages of canalyzation in six columns, for 0%, 100%, and intervals of 25% in between, i.e. 0%, < 25%, < 50%, < 75%, < 100%, 100% (figure 15.2 *Above Right*).

## 15.5 Canalyzing - 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 9.7.1), and offers an option to reset the canalyzation 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)

```

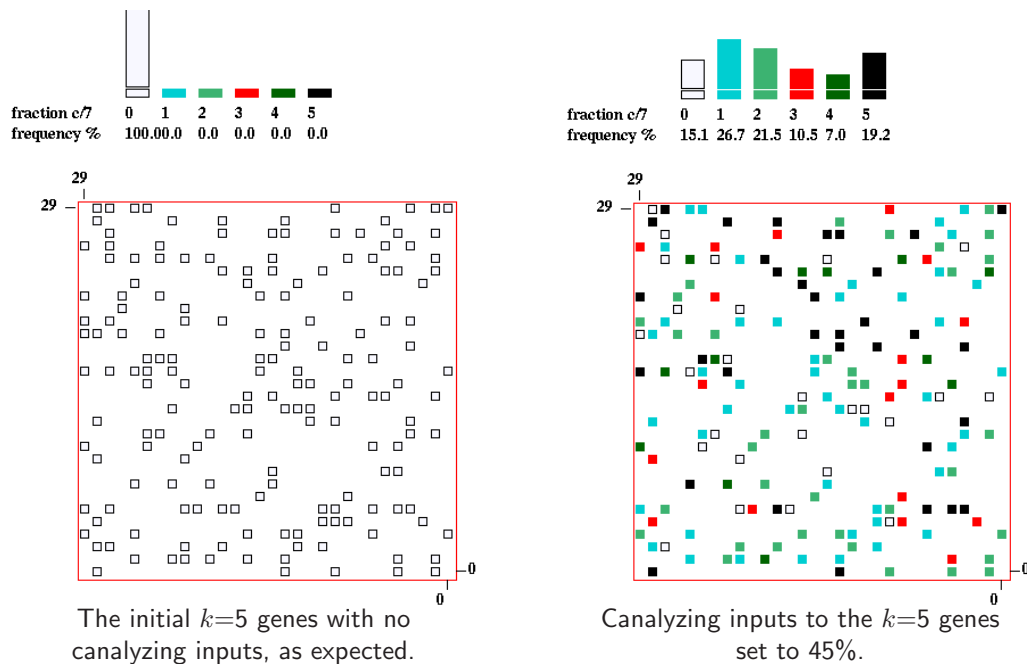


Figure 15.3: Just one  $k$  in a mixed- $k$  network. Canalyzing saturation for two networks with the same (roughly even)  $k$ -mix, 3-7, as shown in section 15.5.  $n=30 \times 30$ . Only the selected  $k=5$  genes are shown. An empty outline indicates no canalyzing inputs and gaps are genes where  $k \neq 5$ . *Left*: The initial network showing  $k=5$  genes with no canalyzing inputs, as expected. *Right*: Canalyzing inputs of the  $k=5$  genes set to 45%. The colors in the network and histogram correspond.

### 15.5.1 Canalyzing for the whole network - mixed- $k$

Enter **return** to set canalyzing for the whole mixed- $k$  network. The next prompt gives the fraction and percentage of the current canalyzing genes and inputs, and asks for the required canalyzing settings. For example, the prompt may appear as follows, in this case for a  $30 \times 30$  network, with a  $k$ -mix of 3 to 7 (figure 15.2),

```
c-inputs=134/4530=3.0% genes=91/900=10.1% (2d network, 30 x 30)
set new % c-inputs, redo-q accept-ret: (values depend on the earlier choices)
```

The reason for the initial canalyzing degree (figure 15.3 *Left*) is that about 23.5% of inputs in randomly assigned  $k=3$  rules are likely to be canalyzing. For  $k=4$  rules the figure is about 1.5% (section 24.9.5).

If, say, **45** (for 45%) is entered. The prompt will reappear with updated information,

```
c-inputs=2034/4530=45.0% genes=794/900=88.2% (2d network, 30 x 30)
set new % c-inputs, redo-q accept-ret: (values depend on the earlier choices)
```

Enter a new canalyzing setting, or enter **return** to accept, the program will continue with options to review network architecture described in chapter 17, or enter **q** to revert to the first canalyzing prompt (section 15.2).

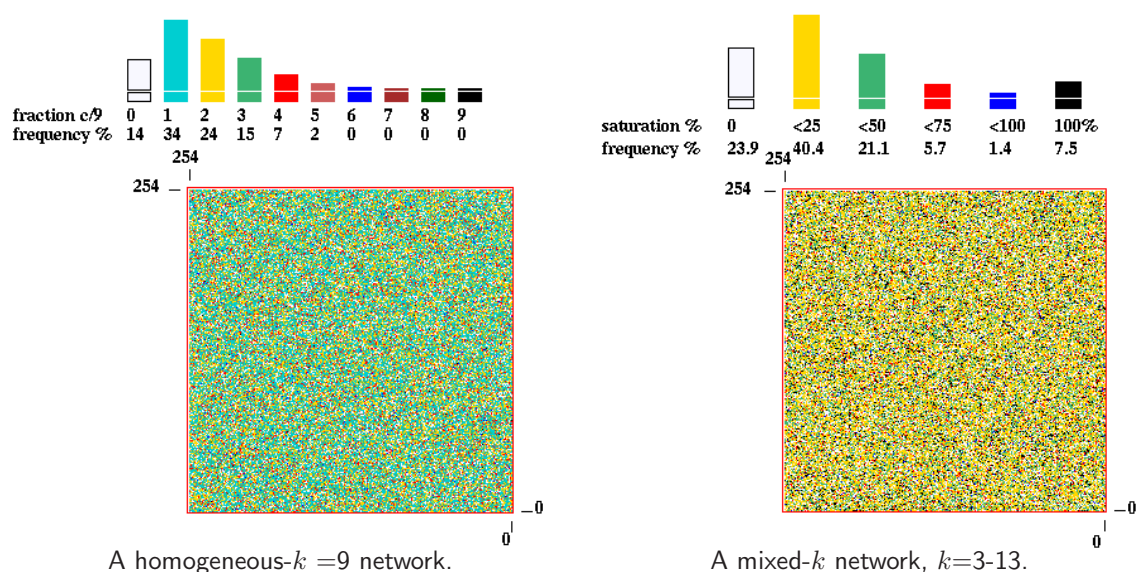


Figure 15.4: Examples of two large networks,  $n=255 \times 255$ , with large  $k$ . The canalizing inputs were set to 20%. Several tries with small increments were required to reach this setting.

At the same time as the prompt appears, a window in the lower right hand corner of the screen gives a 2d graphical display of the degree of canalization of each gene, irrespective of the network's native dimensions (figure 15.3). Above this is a histogram of canalizing saturation for the mixed- $k$  network showing the frequency of different percentages of canalization in six columns, for 0%, 100%, and intervals of 25% in between, i.e. 0%, < 25%, < 50%, < 75%, < 100%, 100% (figure 15.2 Above).

### 15.5.2 Canalizing for a particular $k$ in a mixed- $k$ network

To set canalizing for genes with one specific  $k$  only in a mixed- $k$  network, ignoring the rest, 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 network shown in figure 15.3 Left) the following prompt will appear,

```
k=5 (19.1%) c-inputs=0/860=0.0% genes=0/172=0.0%
set new % c-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 below will reappear with updated information, and the graphic will update – (figure 15.3 Right),

```
k=5 (19.1%) c-inputs=387/860=45.0% genes=146/172=84.9%
set new % c-inputs, redo-q accept-ret:
```

Enter a new canalyzing setting, or enter **return** to accept — the program will continue with options to review network architecture described in chapter 17, or enter **q** to revert to the first canalyzing prompt (15.2).

At the same time as the prompt appears, a window in the lower right hand corner of the screen gives a 2d graphical display of the degree of canalyzation (figure 15.3) showing just the  $k=5$  genes selected. An empty outline indicates no canalyzing inputs and gaps are genes where  $k \neq 5$ .

Above this is the “canalyzing frequency” histogram for just the  $k=5$  genes (figure 15.3 *Above*). This format is employed rather than “canalyzing saturation” because in this example we are dealing with just one  $k \leq 9$ .

---

## 15.6 Canalyzing for large networks, or large $k$

For large networks, or large  $k$ , as in figure 15.4, the algorithm for assigning canalyzing may take some time. Also, the degree of canalyzation achieved may be less than requested. In this case more tries should be made (in sections 15.4, 15.5.1, 15.5.2) to reach the required setting, increasing by small increments.

A top-center window monitors the percentage of canalyzing 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.

---

# Chapter 16

## Setting a single 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 rulemix 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 rulemix: (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) 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 Single rcode prompt examples

For a full rule-table, rcode (not TFO-mode or a  $k$ -mix) a graphic of the binary neighborhood matrix is displayed (sections 13.3.1, 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 ascii-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 ascii-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 ascii-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 ascii-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 ascii-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<sup>1</sup> in section 16.1 are summarized below. More details are given in the rest of this chapter. These options also apply to a rule in a rulemix 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, 32.16.1).

*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 neighborhood  $k+$  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).

**ascii-v** ... to save or load the rule as an ASCII value string, a \*.tbl file, described in section 16.22. This is useful for interchanging the rule between DDLab and alternative software.

*note* ... After the prompts above the program reverts to the first single rule prompt in section 16.1. The prompts below set a rule and the program continues.

**rnd-r** ... to set the rule at random, for a given  $\lambda$  parameter, or proportions of values (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, and many other options. (section 16.4).

**hex-h** ... to set the rule in hexadecimal, in a mini “spread sheet” (section 16.5).

<sup>1</sup>Some of these methods are similar to setting a seed in section 21.2.

- 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 rule-table — not TFO-mode) to show a tcode or kcode as a full rule-table (rcode) which can then be modified.
- iso-i** ... (for rcode only) to set the rule at random but as an isotropic rule, where rotated and reflected neighborhoods (in 1d, 2d and 3d) have the same output.
- rep-p** ... to repeat the last rule (section 16.13).
- load-l** ... to load a rule. from a .rul, .vco, or .tco, file. (section 35.3).  $[v, k]$  in the file must be the same as in the base.

Once a single rule have been set, except after **rnd-r** or in a rulemix “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 simultaneously displayed in the rule window (section 16.19). At any stage enter **q** to backtrack and revise.

---

## 16.3 Setting the rule at random

A rule set at random is unbiased by default, but biases can be applied by two methods, firstly the proportion of non-zero values ( $\lambda$ -parameter), secondly a more explicit bias — the proportions of different values in the rule-table, either as percentages, or as actual numbers of each value. An uneven distribution of values in the rule-table, by either method, is more likely to result in complex dynamics than an unbiased rule-table (figure 33.15).

Enter **r** (the initial default) in section 16.1, to generate a random rule, displayed as a rule-table bit/value string graphic<sup>2</sup> — below the graphic, the rule in decimal (if applicable) and in hexadecimal is also shown. Density-bias is simultaneously displayed in a lower top-right density window (section 16.3.1). Initially there is an equal probability of each value, unless this was reset.

In addition to random biases, various changes and transformations<sup>3</sup> can be made (repeatedly) until **return** is entered to accept the rule.

---

<sup>2</sup>The bits/values options in section 16.4 provide many extra presentation possibilities.

<sup>3</sup>These options are similar to setting a random initial state or seed in section 21.3.

The following subsequent reminder is shown,

**tog-gaps-g**, **rotate-l/r** **another-n** **bias-s/v** **Z-u/d** **comp-m** **back-q** **accept-ret**  
*(Z-u/d for rcode only)*

These options are summarized below — with more detail in the sections indicated.

*options ... what they mean*

- tog-gaps-g** ... to toggle gaps between successive blocks of 8 bits/values. This is the default for binary rules (i.e. between chars) as in figure 16.11.
- exp/contr-e/c** ... enter **e** to expand or **c** to contract the current scale of the rule-table bit/value graphic.
- rotate-l/r** ... enter **l** or **r** to rotate the 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 —  $\lambda$  parameter.
- bias-s/v** ... enter **s** to change the density-bias —  $\lambda$  parameter (section 16.3.1). Enter **v** to set the value-bias, the proportions of different values in the rule-table, either as percentages, or as actual numbers of each value if the rule-table size  $S \leq 255$  (section 16.3.2).
- Z-u/d** ... *(for rcode only)* enter **u** or **d** (or keep the key pressed) to progressively force the  $Z$ -parameter up (towards chaos) or down (towards order), 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**. There is a similar on-the-fly option while running space-time patterns in section 32.5.5.
- comp-m** ... to toggle/transform the rule into its complement. For binary this changes 1s to 0s and vice versa. For  $v > 2$  each value  $a$  is changed to its complement  $a_c = (v - 1) - a$  (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 greater detail below.

### 16.3.1 Random rule density-bias ( $\lambda$ parameter)

Enter **s** in section 16.3 to change the density-bias<sup>4</sup> — the fraction of non-zero values in the rule-table (the  $\lambda$ -parameter [22]) but expressed as a percentage. The initial default is an equal probability of each value, but this may have been reset in section 14.1.2. A lower top-right density window shows information about the density, in this example for rcode  $v5k6$ ,

density 4-1s(exact)=12500/15625=80.000%, bias-80.000%

<sup>4</sup>The methods for the seed density-bias in section 21.3.2 are similar.

where “density 4-1s” indicates the non-zero values, “(exact)” the current method of setting the density as opposed to “(prob)”, 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  $v \geq 3$  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, and when re-setting random rules for space-time patterns with on-the-fly key **r** (section 32.5.1). This is also one of the method to bias a random rule sample for classifying rule space (section 33.2.1).

### 16.3.2 Rule value-bias

	2	65	35							
	3	60	30	10						
	4	55	25	12	8					
value-range <i>v</i>	5	50	25	12	8	5				
	6	45	23	13	10	6	3			
	7	40	21	14	10	7	5	3		
	8	35	20	14	10	8	6	4	3	
		0	1	2	3	4	5	6	7	
		% of each value								

Table 16.1: The default rule value-bias if not reset. The current rule value-bias can be randomly reset when running space-time patterns with on-the-fly key **R**.

The frequency of values randomly distributed in a rule-table, the value-bias, can be set for any distribution. If the frequency was not set previously by the methods below, a default frequency will apply based on percentages of each value as in figure 16.1.

Enter **v** in section 16.3 to set the value-bias, either as whole number percentages, or as actual numbers of each value if the rule-table size  $S \leq 255$ . The following series of prompts are presented in a top-right window<sup>5</sup>,

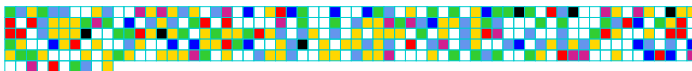


Figure 16.1: Example of a v8k4 kcode according to the default rule value-bias in table 16.1.

<sup>5</sup>The methods for the seed value-bias in section 21.3.3 are similar.

```

if S ≤ 255
bias random rule, %/value-p/v keep-k:
if S > 255
bias random rule, value-p keep-k:
if p was selected
0(100:45 1(55:4 2(51:4 3(47:4 4(43:9 5(34:9 6(25:18
if v was selected
0(120:56 1(64:7 2(57:7 3(50:7 4(43:12 5(31:12 6(19:15

```

Enter **k** to keep the last setting, otherwise enter **p** or **v** — then for each value, from 0 to  $v-2$ , enter the percentage or number — the value for  $v-1$  is set automatically. The availability for successive allocations of each value is shown with a bracket, for example at **3(50:** (for  $v=3$ ) enter a number  $\leq 50$ . If exactly 50 is entered the allocation will halt as complete. Entering a number  $> 50$  or **return** results in allocating 50 divided by the number of remaining values, so entering **return** repeatedly gives an even distribution.

A lower top-right window shows the resulting value-bias distribution for the example above, where “0-7” (for  $v=8$ ) indicates the order of values,

```

if p was selected
rule-bias: percent(100) 0-7: 7 18 9 9 4 4 4 45
if v was selected
rule-bias: numbers of values(120) 0-7: 56 7 7 7 12 12 15 4

```

If set, the value-bias will be maintained for further random rules generated with *another-n* in section 16.3 updating the density window, and when re-setting random rules for space-time patterns with on-the-fly key **R** (section 32.5.1). This is also one of the method to bias a random rule sample for classifying rule-space (section 33.2.1).

### 16.3.3 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,

```

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 [16], 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  $\lambda$  parameter [22].

ld-r ... the  $\lambda$  ratio [31].

P ... the  $P$  parameter [16] — for binary rules ( $v=2$ ) only.

Z ... the  $Z$  parameter [31, 38].

## 16.4 Setting the rule as bits or values

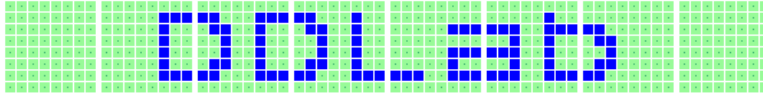


Figure 16.2: Setting bits starting with all 0s and drawing 1s, with default gaps and white divisions.  $v2k9$ ,  $S=512$ .

If **b** is selected in section 16.1, a bit or value graphic 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 successive 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 hand corner of the pattern. The current position on the pattern is indicated by a small flashing cursor, initially top-left, and also in a top-right inset window. Bits/values are set with the keyboard or drawn with the mouse on the rule graphic

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 other bits/values. Alternatively, use the bit/value setting method for fine adjustments to rule-tables set by other methods. Figures 16.2—16.3 illustrate alternative presentations which can be changed on-the-fly.

### 16.4.1 Rules: bits/values reminder window

Bits or values in the rule-table are drawn with the mouse and keyboard<sup>6</sup>. During the procedure, a top-right window, and inset display, gives reminders of various options and current settings (summarized 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: val/draw-(4-0) vert-v move-arrows kcode i=209 val=4 scale=5 rot=1  
 mouse: move-click draw-drag width-w, PScript-P file-F  
 tog: gaps/0color/dots/divs/divcolor-g/^\./i/! exp/contr-e/c xaxis-[/  
 rot-l/r/+/- flip-X comp-m back/cont-q/ret

*options ... what they mean*

**keys: val/draw-(4-0)** ... enter a valid number to select the value/color between 0 and the  $v-1$ . Keep the key pressed to draw a horizontal line.

**vert-v**... keep **v** pressed to draw a vertical line downwards in the selected value/color.

**move-arrows** ... all four arrow keys move the cursor around the rule-table graphic (up/down arrows for multiple rows).

<sup>6</sup>Similar methods apply for the seed in section 21.4.2.

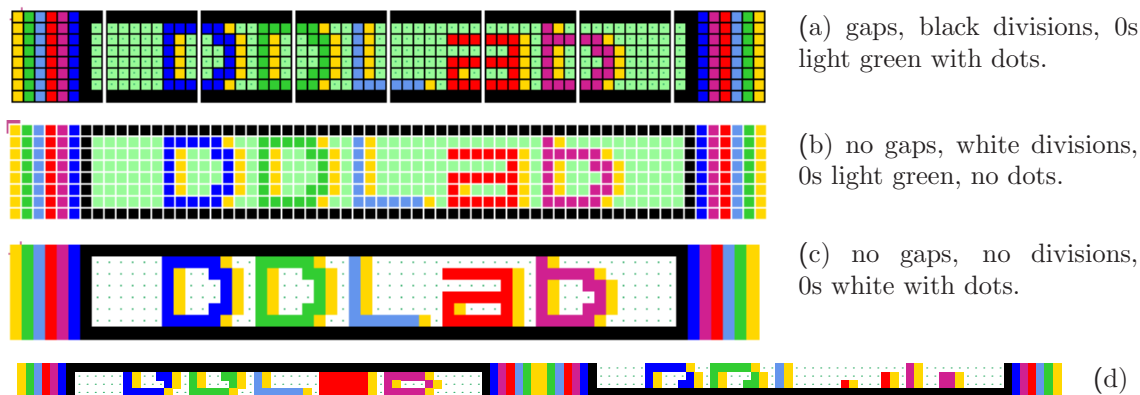


Figure 16.3: Setting values — alternative presentations (a), (b) and (c) of a *v8k3* rcode,  $S=512$ . (d) shows (c) with its x-axis doubled with option ] .

**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 complementary 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.

**PScript-P** ... to save the rule-table bit/value pattern (or a patch) as a vector PostScript image (section 16.4.4).

**file-F** ... to save the rule-table bit/value pattern (or a patch) as a 1d seed file (section 16.4.4).

**tog: gaps-g** ... enter **g** to toggle gaps between successive blocks of 8 bits/values (figure 16.3).

**tog: 0color-^** ... to toggle the zero value color between light green and white (figure 16.3).

**tog: dot-.** ... enter a dot to toggle a dot at the center of each zero cell (figure 16.3).

**tog: divs/divcolor-i/!** ... enter **i** to toggle division lines on/off between the rule-table entries. Enter **!** (exclamation mark) for a 3-way toggle of the division line color — white, black, and light blue (figures 16.1 — 16.3).

**exp/contract-e/c2** ... enter **e** to expand, **c** to contract the scale by one pixel.

**axis-[/]** ... to change the presentation aspect ratio, enter **[** to halve the x-axis, **]** to double the x-axis, the y-axis will change accordingly (figure 16.3)..

**rotate-l/r/+/-** ... enter **l** or **r** to rotate the rule-table left or right by the rotation interval (initially 1) with the edge outputs wrapping around. Enter **+** or **-** to increase or decrease the rotation interval — shown in the current settings inset (section 16.4.2).

**flip-X** ... to flip (reflect) the rule-table.

**comp-m** ... to toggle the rule to its complement. 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/cont-q/ret** ... enter **return** to accept the rule and continue, **q** to backtrack.

## 16.4.2 Rule: bits/values current settings inset

`kcode i=209 val=4 scale=5 rot=1` ← the inset with current settings, for *v5k6* tcode

current settings ... what they mean

`kcode i=` ... the `kcode` index (or the `rcode` or `tcode` index) — the current cursor position according to the rule index, where the bottom-right is 0, and the top-left is  $S-1$ , where  $S$  is the size of the rule-table.

`val=` ... the current drawing value/color.

`scale=` ... the current scale in pixels, which can be expanded and contracted.

`rot=` ... the current interval by which the rule-table can be rotated.

## 16.4.3 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 (section 16.4.2). The cursor is repositioned with the mouse by clicking either the left or right button on the new position<sup>7</sup>, 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 16.2,16.4). 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 complement of the active value (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`

<sup>7</sup>Mouse behavior differs slightly between Linux-like systems and DOS. In Linux the mouse pointer changes direction within the bit/value rule-value 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 pointer is confined within the pattern and clicking the left or right buttons sets values as well as repositioning the flashing cursor.





Figure 16.4: Drawing bits or values on the rcode pattern by dragging the mouse pointer.

*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,  $S=11440$ .

Initially the left button draws the value  $v-1$  and the right draws 0, so for binary 1 and 0. A button press outside the grid area gives changes the reminder window to `outside grid`.

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).

#### 16.4.4 Rules: save as 1d seed or PostScript

The rule-table pattern (or just a patch) can be saved as a vector PostScript image, or as a 1d seed file. Saving as a 1d seed file allows a rule string to seed a basin of attraction, which itself is able to classifying rule space, as proposed by Burraston [6, 7].

Enter *PScript--P* or *file-F* in section 16.4.1 — both options follow the same methods as the 1d seed options in section 21.4.8. One of the following prompts appear in a top-right window,

*for PostScript, PScript-P*

**PostScript KCODE: save all-a, save patch-p:** (*or RCODE, TCODE*)

*for a 1d seed file, file-F*

**RCODE as 1d SEED: save all-a, save patch-p:** (*or TCODE, KCODE*)

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  $v8k3$  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.

#### 16.4.5 Rules: PostScript prompt

When creating a PostScript image of a rule-table, there are a variety of presentation possibilities as illustrated in this chapter, and in section 21.4.10 for a seed where similar methods apply.

Enter **a** (or **p** for part of the rule-table) at the prompt in section 16.4.4 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 PostScript image for 1d, 64x8**  
**symbols-s greyscale-g color-c exit-q (def-c): color-c/C for v=2)**

These options are summarized below (then subsequent options continue),

*options ... what they mean*

**symbols-s** ... to show values as symbols (as in figures 21.10, 21.11 for a seed).

**greyscale-g** ... to show values in greyscale (as in figures 21.10 for a seed).

**color-c** ... (*the default*) to show values in color.

**color-C** ... (*for binary v=2 networks*) by default 1s are colored — blue on a white background and yellow on black background. Enter **C** to instead save 1s in black or white respectively.

Once these options have been selected, the prompt to amend other settings 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,w,b,g):**

Enter changes required or **return** to accept defaults, which follow the current bits/values presentation. The options are summarized below,

*options ... what they mean*

**cellscale** ... enter a new cell width in pixels.

**togdots-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,w,b,g)** ... the initial default division (color) depends on the bits/values presentation, and is shown in the prompt, for example **divs(off)** means that there are no divisions and adjoining cells touch. To change, enter **0** for off, **b** for black, **w** for white, and **g** for light blue/grey. The new choice becomes the default.

Cells with value zero (0color) are initially colored light green in the bits/values presentation (section 16.4), which can be toggled 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 35.3). The default filename is **my\_sPS.ps**, the same as for a 1d space-time pattern. 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 16.1.1 or 16.1.2, 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.

```

0 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 16.5: 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 neighborhoods, such as the *v2k3* “elementary rules” with their well known decimal rule numbers [28, 31] or for  $v=2$  totalistic rules. However the decimal option remains valid so long as the rule-table (rcode, kcode or vcode) is within the limits<sup>8</sup> in table 16.2 which lists the maximum length of the bit/value string  $S$  and the corresponding maximum decimal rule number, for each value-range  $v$ . If **d** is an available option and is selected in section 16.1 or 16.1.2, 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 **return** for a random number which will be displayed. If the number entered is outside the permitted range, the program presents the message,

---

<sup>8</sup>The same limits apply when setting a seed in decimal (table 21.3 section 21.6).

**k=3 rule, enter 0-255 (def-rnd-dec):333 - too big! back-ret:** *(for example)*

Enter **return** to revert to the 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.

$v$	max- $S$	max decimal
2	32	4294967295
3	20	3486784400
4	16	4294967295
5	13	1220703124
6	12	2176782335
7	11	1977326742
8	10	1073741823

Table 16.2: Rules in decimal — rule-table size limitations — the maximum bit/value string  $S$  for each value-range  $v$ , and the corresponding maximum decimal number. These are notional figures because the actual limits also depend on the rule type and neighborhood  $k$ .

## 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 2.3. 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.
- otherwise for kcode or  $v > 2$  one of the majority values is picked at random (see figure 16.7).

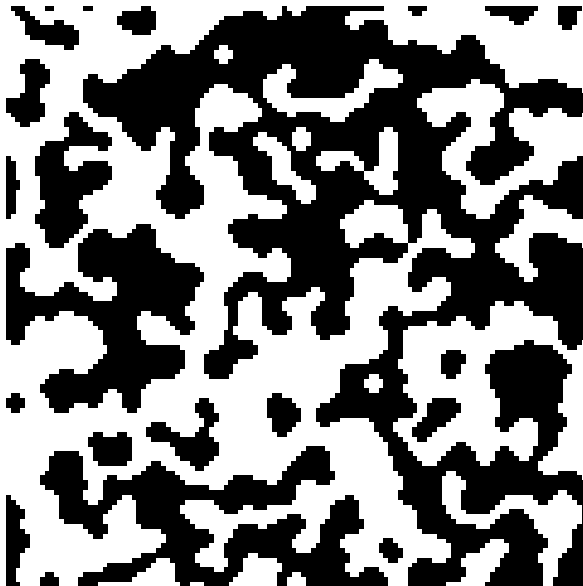


Figure 16.6:

*Above:* Majority rcode  $v2k9$ , shown as a bit pattern of  $S=512$  bits.

*Left:* Space-time snapshot from a random initial state on a  $120 \times 120$  square lattice, where the dynamics has stabilized on an attractor.

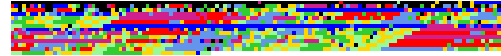
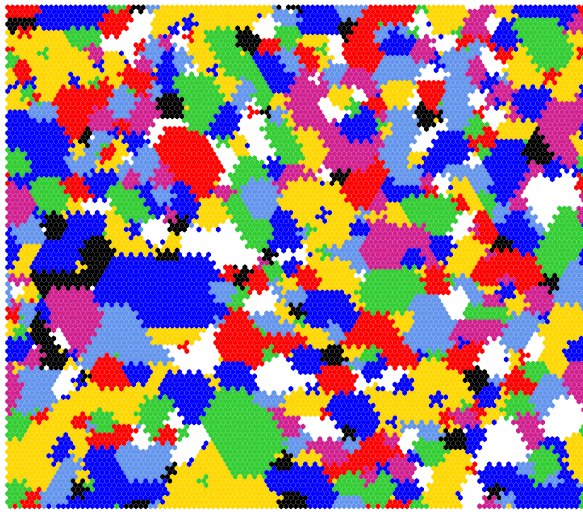


Figure 16.7:

*Above:* Majority kcode  $v8k6$  shown as a value pattern of  $S=1716$  values.

*Left:* Space-time snapshot from a random initial state on a  $120 \times 120$  hexagonal lattice, where the dynamics has stabilized on an attractor.

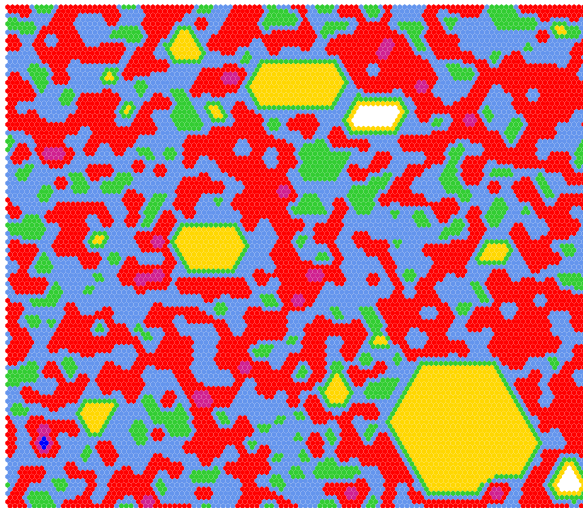


Figure 16.8:

*Above:* Majority tcode  $v8t7$  shown as a value pattern of  $S=50$  values.

*Left:* Space-time snapshot from a random initial state on a  $120 \times 120$  hexagonal lattice, where the dynamics evolves into competing patches.

For tcode, the tcode-table is divided into  $v$  approximately equal sectors, and the values are allocated to the sectors in descending order from the left (see figure 16.8). Once selected, the majority rule will be presented again as bits/values (section 16.4), where it can be reconfirmed or amended.

Figures 16.6 — 16.8, and 4.14. show examples of the majority rule bit/value pattern, and evolved snapshots of 2d spacetime patterns, for rcode, kcode and tcode.

## 16.8 Majority with shifted uniform outputs

*not for tcode — see also section 14.8 for a rulemix*

If  $\mathbf{u}$  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 16.9 for  $v=2$ , and for a rulemix of kcode majority rules,  $v=8$ , in figure 14.4.

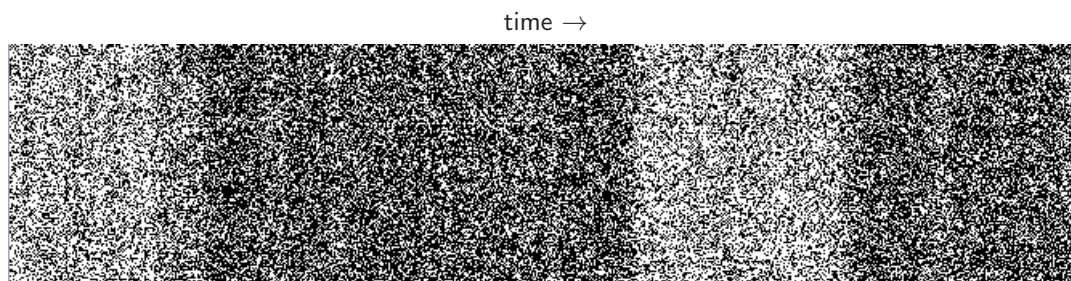


Figure 16.9: Flipped (shifted) majority rcode with random wiring,  $v2k5$ , where uniform neighborhoods (all-0s and all-1s) have their outputs flipped to the opposite color. A 1d space-time pattern ( $n=150$ ) shows bi-stable pattern density, where the duration of the two density regimes is unpredictable. The image shown is rotated by  $90^\circ$  so the time flows from left to right.

## 16.9 Setting Altenberg rules

*not applicable to tcode*

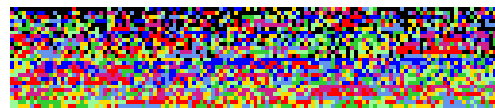
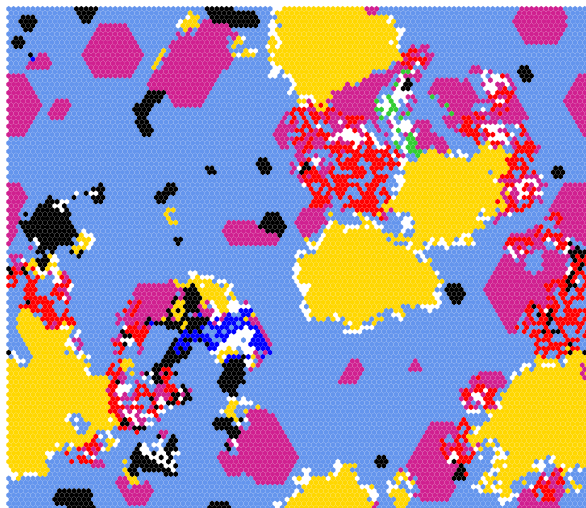


Figure 16.10:

*Above:* Altenberg kcode example  $v8k7$ , shown as a value pattern of  $S=3432$  values.

*Left:* Evolved space-time snapshot from a random initial state on a  $120 \times 120$  hexagonal lattice.



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 16.10, and a 1d space-time pattern in figure 4.10.

## 16.10 The game-of-Life and other Life-like rules — rcode

for rcode and  $k \geq 5$  (for Life in outer-kcode, TFO-mode, see section 14.2.2)

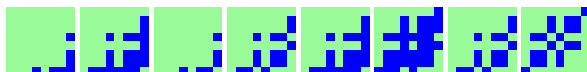



Figure 16.11: Conway's game-of-Life (rule 23/3) shown as a 512 bit rcode.

For  $v2k9$  on a square lattice, and the Moore neighborhood  – John Conway's game-of-Life [10] can be set — see figures 4.11 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-like rules can also be set in outer-kcode in TFO-mode (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 for the full rule-table, rcode.

DDLab defines classical Life as (23/3) following the survival/birth format in Mirek's Celebration<sup>9</sup>. A cell is either alive (1) or dead/empty(0). The first part of (23/3) defines the survival

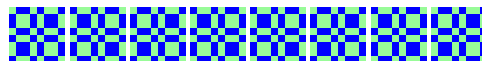
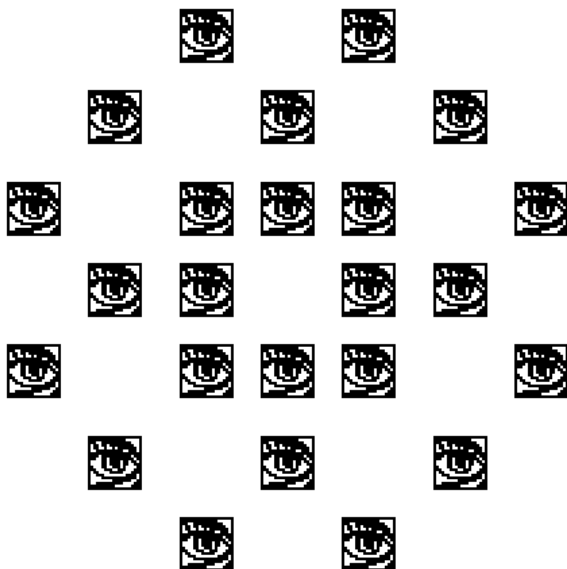


Figure 16.12:

*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.  $222 \times 222$  on a square lattice.

<sup>9</sup>[http://www.mirekw.com/ca/whatis\\_life.html](http://www.mirekw.com/ca/whatis_life.html). The survival/birth (23/3) order is sometimes reversed to birth/survival (3/23) as in <http://www.conwaylife.com>.

of a cell, requiring 2 or 3 live neighbors, the second part of (23/3) defines birth, requiring 3 live neighbors, otherwise the cell is dead (by overcrowding or exposure). The 23/3 notation, when entered in DDLab, is automatically translated into the full lookup-table, rcode (figure 16.11). Any other Life-like rule can be specified in this notation, for example 1357/1357 for Fredkin's replicator in figure 16.12.

### 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 rulemix 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 16.13. To make changes to particular bits/values in Life-like rule-tables, amend bits or values as in section 16.4.

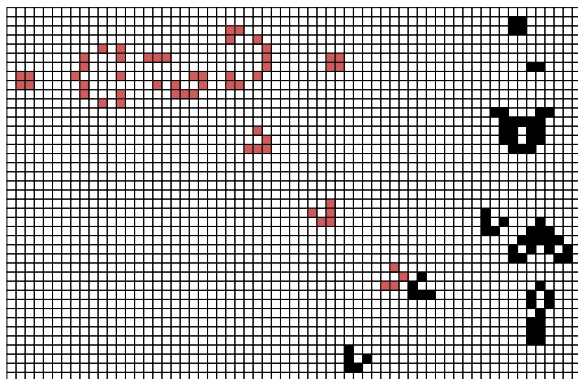


Figure 16.13:

The game-of-Life (23/3) applied to a  $v=3$  CA. The algorithm in DDLab generates an equivalent rcode giving similar dynamics to 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. The seed file is Lguns\_v3.eed.



## 16.11 Setting a chain-rule

*r*code (full rule-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 approaches 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 [47], for example, by running “information” backwards to encrypt, forwards to decrypt as in figure 16.14. The methods apply equally for  $v > 2$ .

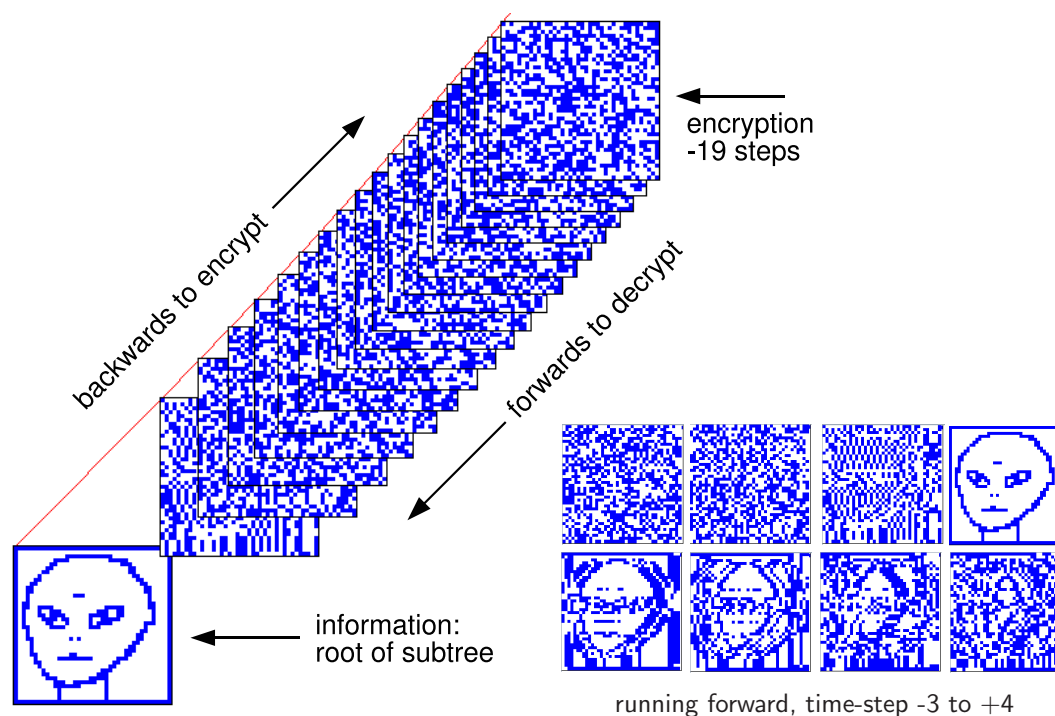


Figure 16.14: *Left*: a subtree applied to encrypt information [47] at its root state (the “alien” seed), set to stop after 19 backward time-steps, where the state reached is the encryption. To decrypt, apply the same rule to run forwards by the same number of time-steps. *Right*: time-steps before and after the alien suddenly pops out, then merges back into chaos. The root state is a 1d bit-pattern, here displayed in 2d ( $n=1600$ ,  $40 \times 40$ ). The alien seed was drawn with the drawing function in DDLab (section 21.4). The seed could also be an ASCII file, or any other form of information. A  $v2k7$  chain-rule (*r*code) 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.

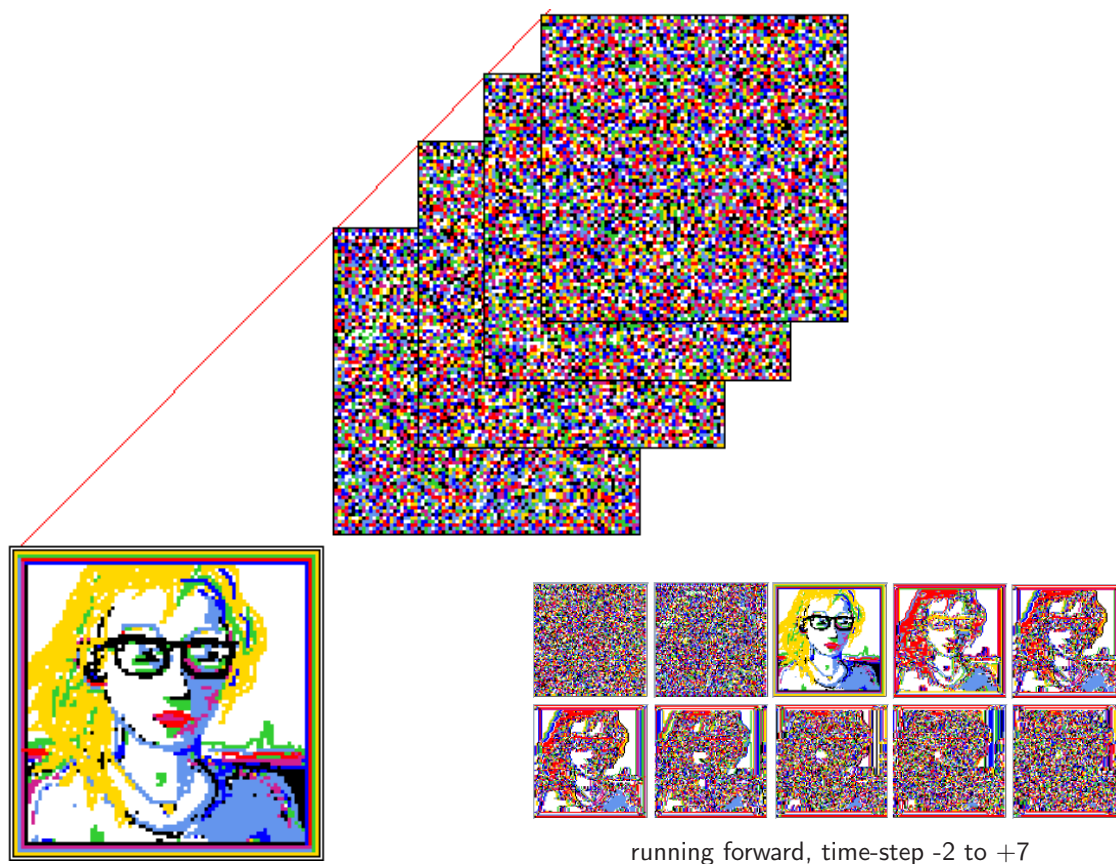


Figure 16.15: *Left*: A 1D pattern is displayed in 2D ( $n=7744$ ,  $88\times 88$ ). The “portrait” was drawn with the drawing function in DDLab. With a  $v=8$ ,  $k=4$  chain-rule constructed at random, and the portrait as the root state, a subtree was generated with the CA reverse algorithm as in Fig. 16.14. The subtree was set to stop after 4 backward time-steps. The state reached is the encryption. To decrypt, run forwards by the same number of time-steps. *Right*: To decrypt, starting from the encrypted state in Fig. 16.15 ( $n=7744$ ,  $88\times 88$ ), the CA with the same  $v=8$  chain-rule was run forward to recover the original image. This figure shows time-steps -2 to +7 to illustrate how the image was scrambled both before and after time step 0.

## 16.12 Setting reaction-diffusion — rcode

for reaction-diffusion by outer-kcode see section 14.2.1

Enter **R** in section 16.1 above to initiate a reaction-diffusion rule with the full rule-table — rcode (sections 13.8, 13.8.2 and figure 13.3). Then set the threshold interval as described in section 13.8.3. Section 14.2.1 describes an alternative method for reaction-diffusion from outer-kcode, which allows greater  $[v, k]$ .

---

## 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 (section 16.15). The file would have been automatically saved when the 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 rulemix, 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 35.3) is presented allowing a single rule to be loaded from a file. The defaults are `myrul_vv.rul` for rcode, `myvco_vv.vco` for kcode, and `mytco_vv.tco` for tcode — for example `myrul_v2.rul`, if the value-range,  $v=2$ . If successfully loaded, the rule is presented again as a bit/value pattern (section 16.4) where it can be reconfirmed or amended. Both the value-range  $v$  and neighborhood size  $k$  in the file must be the same as in the base setup. If not, error messages are displayed in a top top-right window as follows,

```

current-v not equal to file-v
file-v(3) != base-v(2), can't load, not loaded! cont-ret: (for example)

current-k not equal file-k
file-k(3) != base-k(5), can't load, cont-ret: (for example)

```

However, a single rule file can be loaded at a give position in mixed rule network provided  $\text{file-}k \leq \text{base-}k$  — enter **v** from the wiring graphic, section 17.4.

---

## 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<sup>10</sup> starting with 2 leading bytes as 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.

---

<sup>10</sup>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.

See also the file encoding For the wiring/rulemix (section 19.3 and seeds (section 21.9).

$S$  depends on the rule type,  $v$  and  $k$  as follows:

$$\begin{aligned} \text{rcode} \dots S &= v^k \\ \text{kcode} \dots S &= (v+k-1)!/(k!(v-1)!) \\ \text{tcode} \dots S &= v \times k \end{aligned}$$

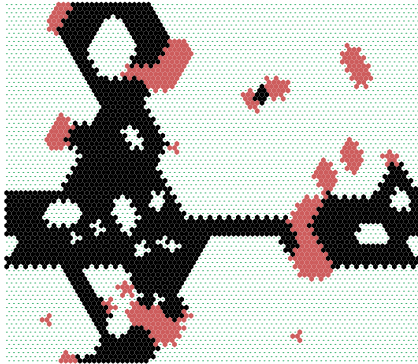
The number of bits required for each rule-table output ( $v_{bits}$ ) is as follows:

$$\begin{aligned} v=2 \dots & 1 \text{ bit } (v_{bits} = 1) \text{ as in the old binary version of DDLab.} \\ v=3 \text{ or } 4 \dots & 2 \text{ bits } (v_{bits} = 2) \\ v=5, 6, 7, 8 \dots & 3 \text{ bits } (v_{bits} = 3) \end{aligned}$$

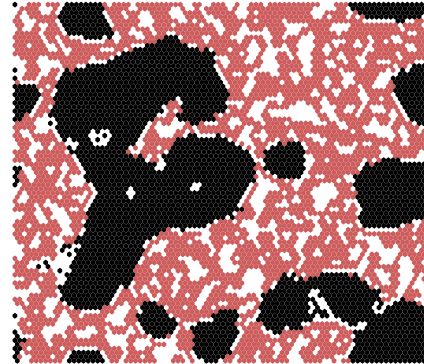
A single rule encoding therefore requires 2 leading bytes plus the rule-table bytes  $R$ , where  $R = \lceil \frac{S \times v_{bits}}{8} \rceil$  bytes, the upper absolute value, minimum 1 byte.

## 16.17 Create a similar kcode with increased $k$

*only for kcode in TFO-mode*



example for  $v3k6$  majority kcode



example for analogous  $v3k7$  kcode

Figure 16.16: The effect of inserting a random string within a  $v3k6$  majority rule to create an analogous  $v3k7$  kcode. The figures show space-time pattern snapshots from random initial states on a  $88 \times 88$  hexagonal lattice. Some aspects of the majority behavior are retained in  $k7$ . Note that there are many possibilities for both examples — the actual kcodes are as shown in this section.

Enter **k** in section 16.1.2 to create and save a kcode with an increased neighborhood ( $k+$ ) 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 16.16. The following top-right prompt is presented,

**create similar to k6 kcode and save, enter greater k(7-25):** (*if the current  $k=6$* )

The two rules are printed in the terminal as shown below — the inserted random string of length 8 is indicated.

*k*-code examples in figure 16.16

```
v_tablecodemax${6}$=27
222222220112001110001111000
v_tablecodemax_new=35 nhood_new=7
22222222011200200022001110001111000}
-----
\
\_ the inserted random string of length 8
```

The new string is automatically saved as the last kcode (`1_vxky.vco`), for example `1_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.

---

## 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). Note that here there is no restriction on the size of the rule-table, for both hex and values strings.

### 16.18.1 Immediate rule data

Immediate rule data will appear in the terminal as in table 16.3 (including the cell index for a rulemix),

*r*code — 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
```

*k*code — for example

```
v3k5 kcodeSize=21
(hex) 00 12 81 51 89 15
(kcode-table:2-0)
001022001110120210111
vfreq=4+9+8=21
```

*t*code — for example

```
v3k7 tcodeSize=15
(hex) 22 06 42 26
(tcode-table:2-0)
202001210020212
vfreq=6+3+6=15
```

Table 16.3: Examples of the immediate rule data in the terminal (xterm) for rcode, kcode and tcode.

### 16.18.2 Immediate rule data for a rulemix by hand

If a rulemix 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 rulemix — 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 16.3.

<u>rcode v3k3</u>	<u>kcode v3k5</u>	<u>tcode v3k7</u>
26: 222 -> 0	0: 0 0 5 -> 1	14 -> 2
25: 221 -> 0	1: 0 1 4 -> 1	13 -> 0
24: 220 -> 2	2: 0 2 3 -> 1	12 -> 2
23: 212 -> 0	3: 0 3 2 -> 0	11 -> 0
22: 211 -> 2	4: 0 4 1 -> 1	10 -> 0
21: 210 -> 2	5: 0 5 0 -> 2	9 -> 1
20: 202 -> 2	6: 1 0 4 -> 0	8 -> 2
19: 201 -> 2	7: 1 1 3 -> 2	7 -> 1
18: 200 -> 1	8: 1 2 2 -> 1	6 -> 0
17: 122 -> 1	9: 1 3 1 -> 0	5 -> 0
16: 121 -> 0	10: 1 4 0 -> 1	4 -> 2
15: 120 -> 0	11: 2 0 3 -> 1	3 -> 0
14: 112 -> 0	12: 2 1 2 -> 1	2 -> 2
13: 111 -> 1	13: 2 2 1 -> 0	1 -> 1
12: 110 -> 2	14: 2 3 0 -> 0	0 -> 2
11: 102 -> 1	15: 3 0 2 -> 2	outputs
10: 101 -> 2	16: 3 1 1 -> 2	totals=index
9: 100 -> 0	17: 3 2 0 -> 0	
8: 022 -> 2	18: 4 0 1 -> 1	
7: 021 -> 0	19: 4 1 0 -> 0	
6: 020 -> 1	20: 5 0 0 -> 0	
5: 012 -> 1	outputs	
4: 011 -> 1	neighborhoods	
3: 010 -> 0	index	
2: 002 -> 1		
1: 001 -> 0		
0: 000 -> 1		
outputs		
neighborhoods		
index		

Table 16.4: Examples of the vertical layout for rcode, kcode and tcode, with explanatory notes, for the same rules as table 16.3.

### 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 which applies only if  $v=3$  — enter **m**. The matrix layout is clarified in section 13.6.2. The results in the terminal are shown in table 16.5,

<i>kcode horizontal layout v3k5</i>	<i>kcode matrix layout v3k5</i>
2:544333222211111000000	1 1 1 0 1 2
1:010210321043210543210 --neighborhoods	0 2 1 0 1
0:001012012301234012345	1 1 0 0
	2 2 0
001022001110120210111 --outputs	1 0
	0

Table 16.5: Examples of (*left*) horizontal, and (*right*) matrix, layout for the same kcode as table 16.4.

### 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 16.6 in some of the same layouts as can be seen in the terminal. Figure 16.18 shows an example of the equivalent space-time patterns.

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 16.6: 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 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  $\lambda$ ,  $P$ , 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 16.17, and decoded in section 16.19.1. As much of the hex rule-table as will fit will be shown<sup>11</sup>.

---

<sup>11</sup>Data in the rule window can be printed to the terminal (section 16.18) with no restriction on rule-table size.

*full rule-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 rule-table setting kcode, so both rcode and kcode are shown*

```

v4k5 rcode(hex)e7b345f7 b3cc31c445314f5ef7c45ecab3cc31c4cc11cc1131cc3e48c411481145314f5e31cc3e484f3effee5e48ee89f7
kcode(hex)= e7 cd 73 04 fa 47 1e 1f 9d 24 7b 9b 28 44
1d size=150 ld=0.687 ld-r=0.915 zl=0.348 zr=0.348 Z=0.347656 C=0/5

```

*kcode in TFO-mode and 2d*

```

v8k5 kcode (TFO)
kcode(hex)= 55828f 88e714de9f1aaf799dd2e3223c150ef817a809f86bcb42d06530a0371285430a6cc2353acc49
2d size=222x222

```

Figure 16.17: Rule window examples, images grabbed from the screen.

### 16.19.1 Decoding the rule window

The abbreviated headings in the first example of the rule window (figure 16.17) are explained below,

```

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 rule-table (rcode) only*

**ld** ... the  $\lambda$  parameter [22].

**ld-r** ... the  $\lambda$  ratio [31].

**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) — here there is 1 canalizing input, at neighborhood index 0.

**Post=A[0]i** ... Post-function data, shown only if the rule qualifies (section 14.12).



## 16.20 Complementary values

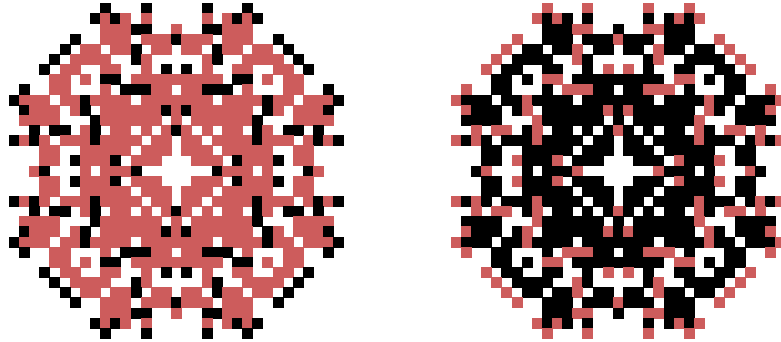


Figure 16.18: Space-time snapshots of 2d CA, with two equivalent  $v3k6$  kcodes, with swapped values 1 and 2 (red and black). The initial state was a single central cell on a  $40 \times 40$  square lattice. The snapshot was taken at time-step 54.

Options to complement values apply to both rules and seeds, for example **comp-m** in sections 16.3, 16.4, 18.5.1, 21.4 and 21.4.3.

Transforming a value into its complement is simple for binary ( $v=2$ ) – change 1s to 0s and vice versa. For *any* value-range  $v$  the “complement” of a value in DDLab is defined as follows: each value  $a$  is changed to its complement  $a_c$  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 complement 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
\_/	\_ \_ \_ /	\ \_ \_ / \_ \_ \_ /	\ \_ \_ \_ / \_ \_ \_ \_ /	\ \_ \_ \_ / \_ \_ \_ \_ / \_ \_ \_ \_ /	\ \_ \_ \_ / \_ \_ \_ \_ / \_ \_ \_ \_ / \_ \_ \_ \_ /	\ \_ \_ \_ / \_ \_ \_ \_ / \_ \_ \_ \_ / \_ \_ \_ \_ / \_ \_ \_ \_ /

## 16.21 Transforming the single rule

*not in TFO-mode*

For full rule-tables (rcode) only, including kcode and tcode expressed as rcode, single rules that have been set can be transformed in various ways. The the rule’s equivalence class and rule cluster [31] can also be shown in the terminal. 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 (for example)
save/prtx: rcode/kcode-s/S/x/X EquivClass-E RuleCluster-R:
```

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 interrupt options (section 32.16.1) and attractor basin complete options (section 30.5).

---

## 16.22 Saving/Loading a rule as an ASCII string

Enter *ascii-v* at the rule prompt in sections 16.1.1 or 16.1.2 to save or load the rule-table as an ASCII value string, a \*.tbl file, (see Filing chapter 35). This is different from a binary rule file (section 21.9), and is useful for interchanging rules between DDLab and alternative software.

The following top-right prompt is presented, for example,

**ASCII rcode-table (v=4 k=3 S=64) load/save-l/s:** *(or kcode, or tcode)*

When loading, if file-*v* is not equal to base-*v*, the following top-right notice is displayed,

**file-v(8) != base-v(5), abort-q, load anyway-ret:** *(for example)*

and if file-size is not equal to base-size, the following top-right notice is displayed,

**file-size(512) != base-size(81), abort-q, load anyway-ret:** *(for example)*

In either case, enter **q** to abort loading, or **return** to continue loading. Although the loading prompt warns of a conflict between the file and base (value-range *v*, or table-size *S*) with an option to abort, any ASCII file within DDLab's naming constraints (section 35.1) with \*.tbl extension can nevertheless be loaded, so file-*k* can differ from base-*k* and there is no distinction between rcode, kcode and tcode. As much or as little of a \*.tbl file as will fit can be loaded, starting at rule-table index zero. Any file-*v* greater than maxbase-*v* will be reduced to maxbase-*v*.

### 16.22.1 ASCII rule encoding

ASCII encoding for a rule \*.tbl file (the same as for a seed\*.tbl file)<sup>12</sup> is as follows: the first byte gives the value-range *v*, then a byte for each value at index 0 to *S*-1. For example, the ASCII file, and the rule-table itself, for the majority rcode, *v*=4, *k*=3, *S*=64, are shown below

*the ASCII file — 1+64 bytes*

40000011002230333011311112121012301220121222203230023111333233333

*the rcode-table itself — 64 bytes, shown numerically and graphically*

333323331113200323022221210221032101212111131103330322001100000




---

<sup>12</sup>The ASCII rule and seed encoding (\*.tbl and \*.see are the same, so the a seed can be loaded into a rule and vice-versa, provided the file-name extension is changed accordingly.

# 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 spreadsheet 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.

Alternatively 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 unravelled 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 predefined parts of the network. These are very flexible methods, and for RBN/DDN its usually easier to set up a suitable dummy network initially, then tailor it here.

For any wiring graphic, new functions make use of the mouse pointer (section 17.5) — a mouse click will reposition the active cell — the last two clicks will define the default corners of a block, making it easier to explore and amend network wiring.

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 a 3d isometric (2d+3d), where the 2d view shows horizontal slices through the 3d network — “levels” stacked above each other, and provides the surface for the mouse pointer.

---

### 17.1 The network architecture prompt

*see also “Reviewing wiring” section 12.7*

Once the wiring (chapters 11 or 12) has been set, and subsequently the rule or rules (chapters 14 or 16), and possibly transformed (chapter 18), a top-right network architecture prompt is presented. The exact wording and options offered depend on previous settings. The prompt can also be

activated at later stages in DDLab, while drawing space-time patterns (section 32.16) or attractor basins (sections 30.4, 30.5, 30.5.1).

The prompt (examples in section 12.7) starts as follows, showing default network sizes,

```
1d network (n=150), ... (for a 1d network)
2d network (40x40), ... (for a 2d network)
3d network (9x9x9), ... (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 rcodes set, the prompt appears as follows,

```
1d network (n=150), review/revise/learn, wiring and rcodes
graph-g, matrix: revise-m view-M prtx-Mp
graphic: 1d:timesteps-1 circle-c 2d-2:
```

*options ... what they mean*

**graph-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** ... for the network architecture in a spread-sheet type format, the wiring matrix, where the wiring can be changed.

**view-M** ... to view the wiring matrix and save matrix image as a vector PostScript file 17.2.1.

**prtx-Mp** ... to view as above, but also to print the wiring matrix 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 17.6, applies also to 2d and 3d.

**circle-c** ... enter **c** for a 1d wiring graphic arranged in a circle, section 17.6, applies also to 2d and 3d.

**2d-2** ... enter **2** for a 2d wiring graphic, section 17.7, applies also to 1d.

**2d+3d-3** ... enter **3** for the wiring graphic in 3d, and simultaneously in 2d in successive horizontal slices, section 17.8.

## 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 its image as a vector PostScript file; **m** to change the wiring. The rules in a rulemix (if set) are also shown, as much as will fit; 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,  $K$  ( $k-1, \dots, 0$ ). Rows give the cell network position,  $N$  ( $n-1, \dots, 0$ ). 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  $x$ , ( $n-1, \dots, 0$ ), 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 — sections 10.2.2 and 10.2.3 explain how to convert  $x$  to 2d or 3d coordinates, and vice versa. A column to the left of the cell index 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 eb 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 ed ce	
5	10:								8	13	12	4	4	2	e4 ef d7 0e 57 47 70 30	
7	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 e2 3f	
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 ee 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.  $k-1, \dots, 0$  indexes columns,  $n-1, \dots, 0$  indexes rows. The column on the left shows the “out-degree” of each cell, the number of wires leaving it, also shown as a histogram. For a rulemix only (if set), rules are shown in hex, as much as will fit, in the right column. For  $k \leq 3$ , decimal rules are added. If the matrix was set with **m** in section 17.1, the wiring can be set by hand as in a spread-sheet (section 12.6) and a rule in the active cell also appears in the rule window (section 16.19). If the matrix was set with **M** in section 17.1, the matrix cannot be amended, but 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 also amend) together with the following top-right prompt,

*if the matrix fits one window*

**PScript-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 PScript-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.
- PScript-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** to 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 12.6. In both cases, the following top-right reminder is presented,

**hand wire/revise:jump-j** (for a 2d network, 40x40)

**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, including use of the mouse pointer/click (section 17.5) to reposition the active cell, and define the default corners of a block. 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 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 predefined block of cells.

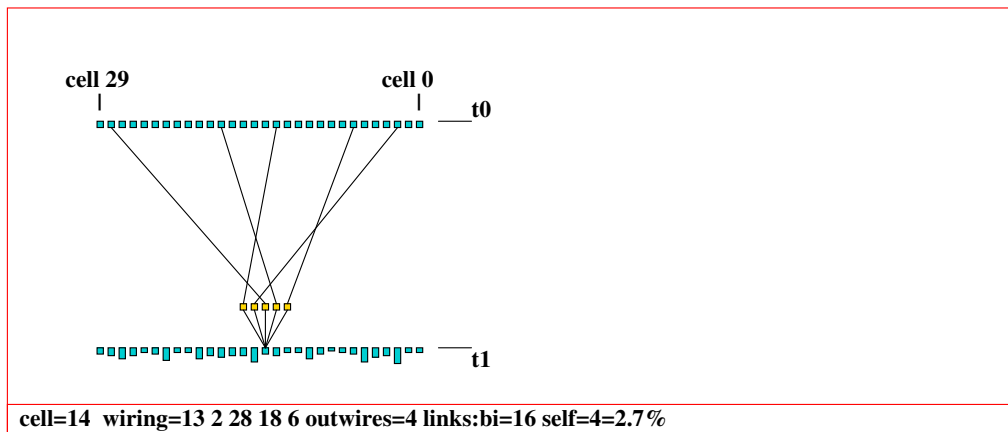


Figure 17.2: The 1d wiring graphic showing the wiring between successive time-steps,  $k=5$ ,  $n=30$ . Each cell's “outwires” are represented by the heights at time-step  $t_1$ . For data decode see section 17.9

## 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<sup>1</sup>. The first line of the reminder shows the network dimensions and size. If rule/s have not been set, the first line reads **wiring only** instead of **wiring/rules** and some options will be missing. **change k** only applies for rcode combined with mixed-*k*, and **kill-K** only for mixed-*k*.

As a consequence of larger network sizes in 2d and 3d, the new default when defining a block is to show just its edges, to speed up the graphics presentation for large blocks, but this can be toggled to show the block in full as before with **edges-G** (sections 17.7.5 and 17.8.5). If a block is active, the prompts show **tog:block-g** (instead of **tog:all-g**) and the block's start-end (for 1d) or opposite corner coordinates (for 2d or 3d) are shown instead of the coordinates of the active cell, for example **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 — for 1d (can work for 2d or 3d)*

**1d (n=150) wiring/rules: move-arrows jump-j/mouse-click**  
**one cell: right/left arrows, 15 cells: up/down arrows, PScript-P**  
**block-b tog:all-g pseudo-p, in/out-i/o avZ-z hilght/Lrn-l/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 — for 2d (can also work for 1d)*

**2d (40x40) wiring/rules: move-arrows, jump-j/mouse-click**  
**exp/contr-e/c up/down-u/d redraw-T PScript-P toghex-x**  
**block-b tog:all-g cell-links(1)-n pseudo-p index-i avZ-z hilght/Lrn-l/L**  
**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)/Out/Both-I/O/B reset-q cont-ret**

*3d wiring graphic — with an active block*

**3d (20x20x20) wiring/rules: move-arrows, levels up/down-[/] jump-j/mouse-click**  
**exp/contr-e/c/E/C up/down-u/d grid3d-T hide-W PScript-P**  
**block-b tog:block-g edges-G block-links(4)-n pseudo-p index-i avZ-z hilght/Lrn-l/L**  
**rewire 3d block 2,2,2-7,7,7: hand-h rnd-r/w/R special-s local:1d-1 2d-2 3d-3**  
**change k=6(max 13)-k kill-K rule: Save/rev/trans-S/v/t Derrida-D**  
**file/data-f hist:In(k)/Out/Both-I/O/B, reset-q cont-ret**

<sup>1</sup>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 activate as soon as the key is pressed (without **return**) — some have secondary prompts. Below is just a brief summary roughly 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- mouse-click** ... use the mouse pointer and click to reposition the active cell — the last two clicks define the default corners of a block (section 17.5). For 1d, 2d and 3d — sections 17.6.2, 17.7.2 and 17.8.2.

**move-arrows jump-j** ... use the arrow keys to move around the network and reposition the active cell. (or one level in 3d) or enter **j** to jump to a particular cell index. For 1d, 2d and 3d — sections 17.6.2, 17.7.2 and 17.8.2.

**one cells: right/left arrows** ... (*1d graphic and circle only*) to move left or right. (section 17.6.2).

**15 cells: up/down-u/d** ... (*1d graphic and circle only*) to jump 1/10th left or right (section 17.6.2).

**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 17.8.9). Also works for the 2d wiring graphic (17.7.8).

**levels up/down-[/]** ... (*3d graphic only*) use the square braces, “[” for up, “]” for down, to move between levels (section 17.8.2).

**exp/contr-e/c** ... (*2d and 3d graphic only*) to expand or contract a 2d wiring graphic (section 17.7.7), or the 2d version of the 3d wiring graphic (section 17.8.8).

**exp/contr-E/C** ... (*3d graphic only*) to expand or contract the 3d isometric (section 17.8.8).

**redraw-T** ... (*2d graphic only*) to redraw the current wiring graphic, in case its covered by the wiring graphic reminder itself (section 17.4).

**grid3d-T** ... (*3d graphic only*) to toggle grid lines on the 3d isometric (section 17.8.7).

**\*PScript-P** ... to redraw and create a vector PostScript file of the current wiring graphic (section 17.9.14).

**toghex-x** ... (*2d graphic only*) to toggle between square and hex layout.

**block-b** ... define a block of cells in 1d, 2d or 3d, for resetting the wiring or rules (sections 17.6.3, 17.7.5, 17.8.5).

tog: other toggle options ...

**all-g** ... (*block inactive*) toggle showing the active cell on its own, or the whole network (with caution for large networks). Combine with **n** to see wiring.

**block-g** ... (*block active*) toggle the block and its connections, to make visible or invisible (but still active). Combine with **n** to see wiring.

**edges-G** ... (*2d and 3d graphic only*) toggle between block edges (the default) and a full block. Combine with **n** to see wiring.

- cell-links(1)-n** ... (2d and 3d graphic only — block inactive) a 5-way toggle for alternative presentations of wiring relative to either the active cell (or the whole network if set with **all-g** above). For the active cell these options may be used with **pseudo-p** below. (sections 17.7.3, 17.8.3).
- block-links(4)-n** ... (2d and 3d graphic only — block active) a 5-way toggle for alternative presentations of wiring for the block, either just its edges or the full block set with **edges-G**. (sections 17.7.3, 17.8.3).
- pseudo-p** ... (for the active cell) toggle to showing or omit the pseudo-neighborhood.
- index-i** ... (2d and 3d graphic only) Toggle the index numbers in the pseudo-neighborhood. The numbers shown up if the scale is large enough.
- miscellaneous options ...
- in/out-i/o** ... (1d and circle graphic only) show the indirect inputs and outputs of past time-steps, the “degrees of separation” between cells (sections 17.6.6, 17.6.7).
- \*avZ-z** ... (rcode-mix and/or k-mix) to calculate the average and weighted average  $Z$  and  $\lambda$  parameters (section 17.9.2).
- \*hilight/learn-l/L** ... (rcode only) enter **L** to implement the learning, forgetting, highlighting functions. Enter **l** for only highlighting nodes in attractor basins (chapter 34). When interrupting space-time patterns the option is **learn-L** — highlighting does not apply.
- rewire: options ... shows the coordinates of the cell, or block if active.
- untangle-u** ... (1d and circle graphic only) “untangle” the wiring of the cell, block, or the whole network, and reset the rules for equivalent dynamics (section 17.6.8).
- \*hand-h** ... rewire the active cell by hand (section 17.9.4).
- \*rnd-r** ... to randomly rewire the active cell or block, respecting any biases set in **special-s** below. Otherwise, for 2d and 3d, the maximum reach is limited by the shortest axis (section 17.9.5).
- \*rnd-w** ... as for **rnd-R** above, but if a block is active randomly rewire cells *outside* the active block (section 17.9.5).
- \*rnd-R** ... For homogeneous- $k$ , **rnd-R** gives unbiased random wiring, either to a single cell or block — for 2d and 3d there is no limit on the reach as in **rnd-r** or **rnd-w**. For mixed- $k$ , **rnd-R** will bias the wiring according to the  $k$  distribution in the  $k$ -mix, allowing a power-law distribution of outputs as well as inputs described in section 9.7.2 (section 17.9.5).
- special-s** ... set “special” wiring biases, which can then be applied to the active cell, or block (inside or outside) with **rnd-r** or **rnd-w** above (section 17.9.6).
- \*local:1d-1 2d-2 3d-3** ... set local CA wiring for the cell or block (section 17.9.7).

**\*change k=8(max 13)-k ...** (*r*code-*m*ix set) change  $k$  up or down for the cell or block, up to  $k_{Lim}$  (section 7.2).

**\*kill-K ...** (*r*ule-*m*ix set) kill or neutralize a cell by cutting all its links, both inputs and outputs, except for one input to itself. The rule is also reset for effective  $k=0$  (section 17.9.9).

**del-d ...** (*1d networks, 1d or circle graphic only, k-mix, rcode set, in main prompt sequence, active cell>0, not in TFO-mode*) delete a cell from the network, and shorten the network by one (section 17.6.9).

rule:Save/rev/trans-S/v/t 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 17.9.10).

**\*trans-t ...** transform the rule or rules, set canalizing inputs (section 17.9.11).

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 17.9. Other options apply more specifically, or uniquely, to a particular dimension — these options are described further in sections 17.6 for 1d, 17.7 for 2d, and 17.8 for 3d.

## 17.5 Wiring graphic, mouse pointer/click

New functions in the wiring graphic allow a mouse pointer/click to reposition the active cell, and the last two clicks to define the default corners of a block. The active cell can still be moved around with the arrow keys, or “jumped”, as before, but the mouse click method provides an additional intuitive way to explore and amend network wiring. The pointer/click applies to all wiring graphics, 1d time-steps or 1d circle layout (section 17.6), 1d as 2d or 2d square or hex (section 17.7), and 3d where the pointer moves on the 2d version of the 3d network (section 17.8).

In Linux-like systems the usual North East pointer will change direction to North West if within the active zone of the wiring graphic — for DOS the pointer will be confined within the active zone. While inside the active zone, a small top center window gives a continual readout of the network coordinates, for example for a 1d, 2d or 3d graphic,

i=1268

i,j=34,17

i,j,h=6,5,10

## 17.6 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 predefined 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 mouse pointer clicks or arrow keys, and it’s possible to “jump” to a new location. The rule for the active cell appears in the rule window described in section **16.19**.

Sections **17.6.1**–**17.6.9** below explain some of the options specifically for 1d in more detail that were summarized in section **17.4.1**.

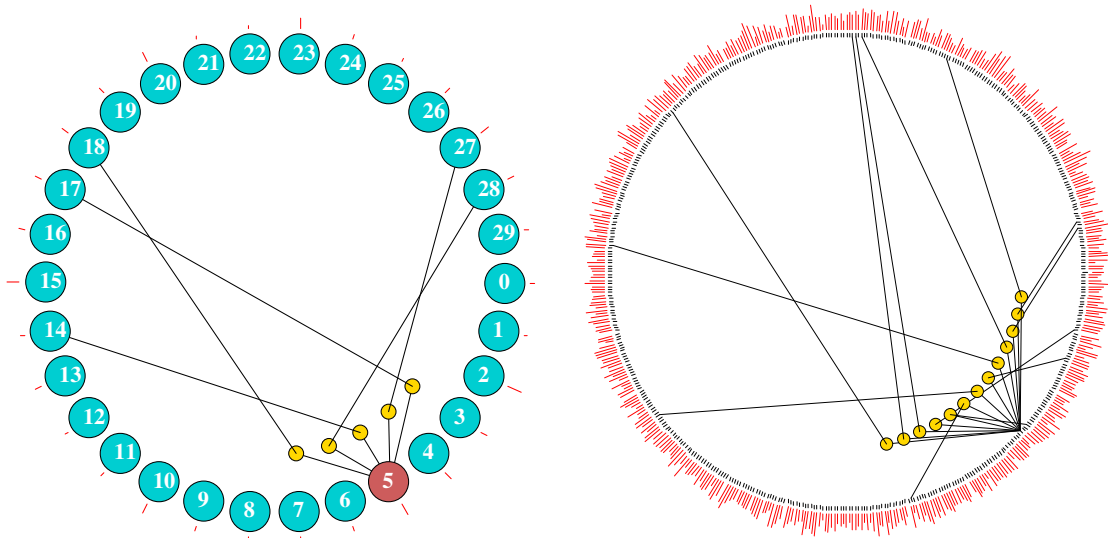


Figure 17.3: Examples of the 1d circle wiring graphic. *Left*: the same network as in figure **17.2**,  $k=5$ ,  $n=30$ . *Right*:  $k=13$ ,  $n=500$ . The “active” cell is repositioned by mouse pointer clicks or moved by the arrow keys. Each cell’s “outwires” are represented by the length of the red 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.

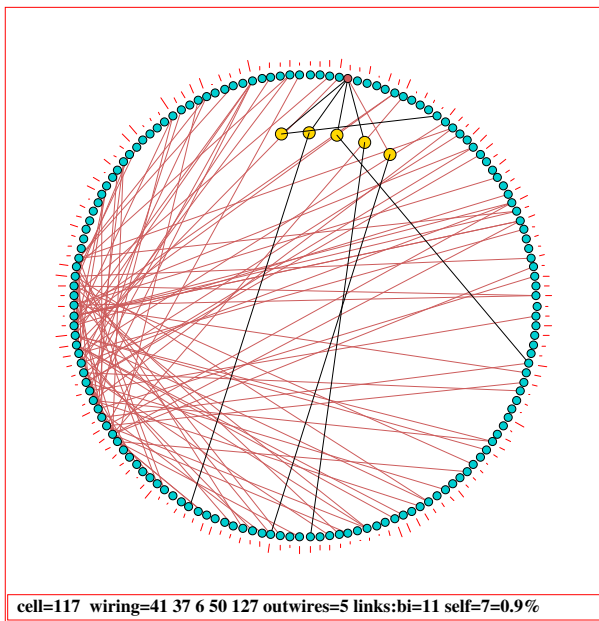
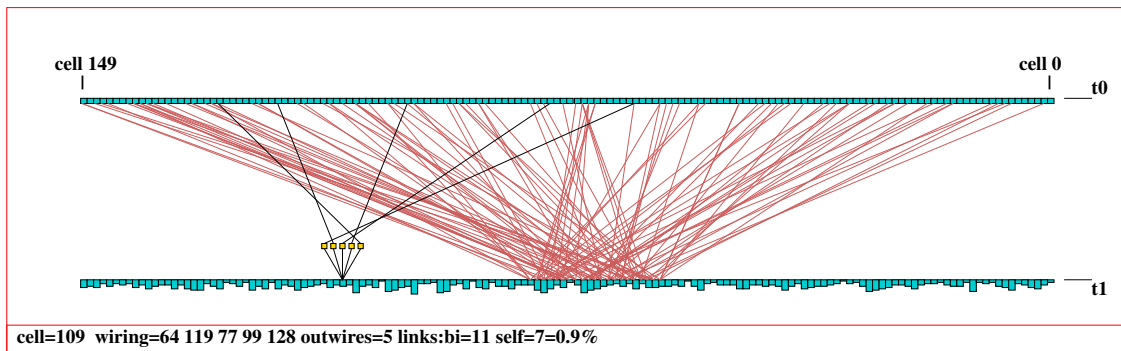


Figure 17.4: The 1d wiring graphic, showing wiring to a block of 11 cells,  $k=5$ ,  $n=150$ . *Above:* As time-steps. *Left:* As a circle. The block was defined from cell 60–80 and has random wiring set with key  $r$ . The “active cell” (109) is still visible, and can be moved with a mouse click or arrow keys usual.

### 17.6.1 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 nonlocal 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 17.9.1 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.

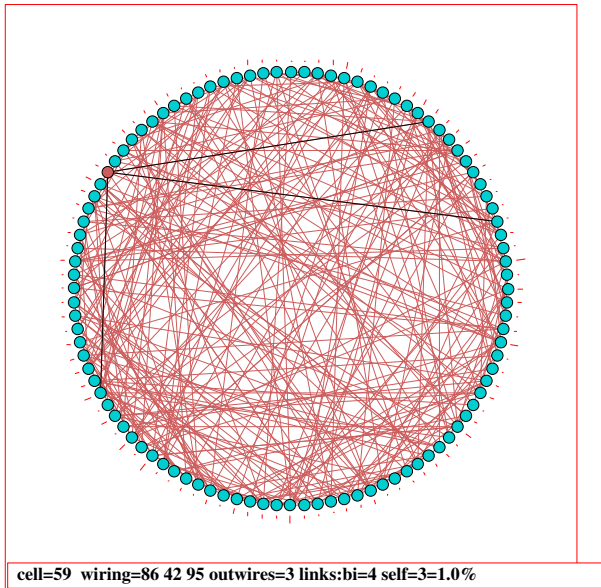
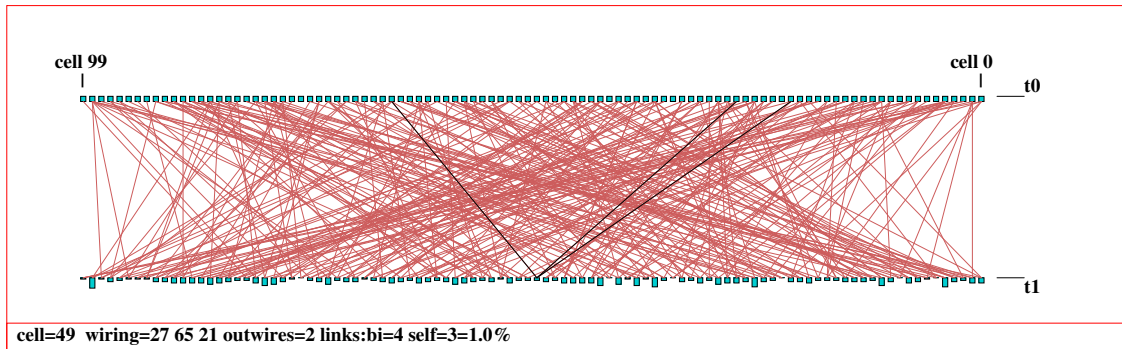


Figure 17.5: The 1d wiring graphic, showing the wiring of the whole network,  $k=3$ ,  $n=100$ . If a block is not set,  $g$  toggles the full wiring on and off. In these examples, the pseudo-neighborhood was toggled off with  $p$  to show just the (black) direct links to the active cell. Above: As time-steps. Left: As a circle.

### 17.6.2 Moving or jumping between cells, 1d

To move the active cell, position the pointer and click any mouse button, or use the arrow keys. The left and right arrow keys move by one cell, the up and down arrow keys move 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)

### 17.6.3 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, no block and single cell-def:

(then, if 1 is selected ...)

1d block (0-150), first(def 60):      second(def 80):

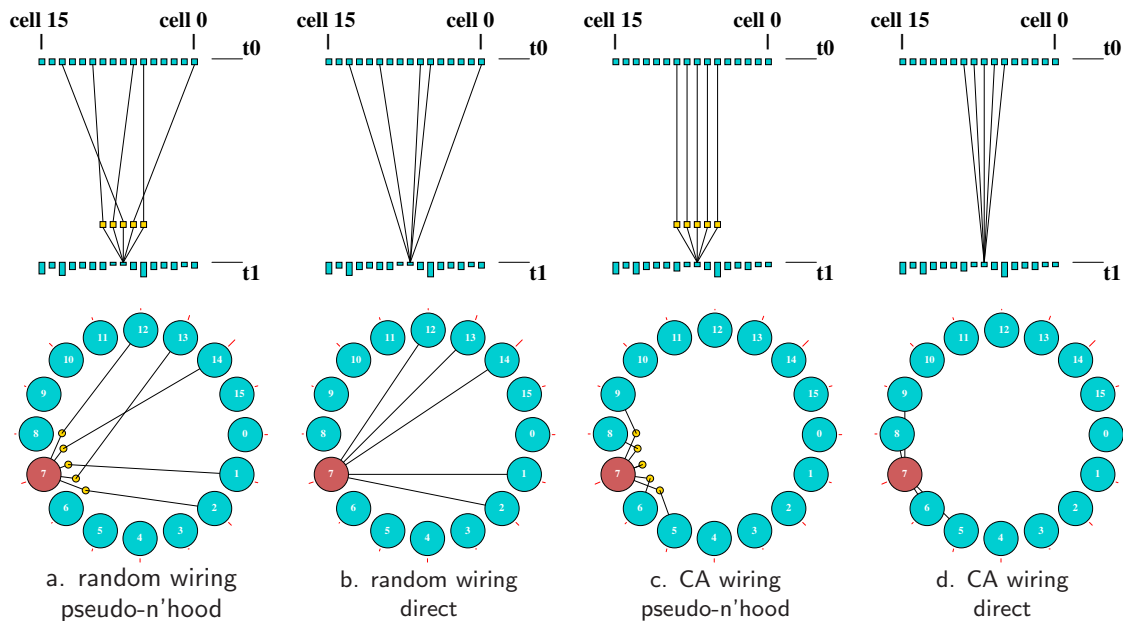


Figure 17.6: Examples of the 1d wiring graphic,  $k=5$ ,  $n=20$ . (a) and (c) include the pseudo-neighborhood, (b) and (d) show just direct wiring, toggle between with **p**. (a) and (b) have random wiring, (c) and (d) have their wiring changed to 1d CA by entering **1**. The same wiring inputs to cell 7 are shown: 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 first and second cell indexes delimiting the block. The defaults correspond to the indexes of the last two mouse clicks. Enter **return** to accept a default, or enter any required index.

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 17.7.5

An active block is indicated in the wiring graphic reminder (section 17.4) for example,

```
rewire 1d block 60-80: ...
```

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 — rcode-mix set)
```

... 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 (section 17.9.10).

```
... rule:Save/rev/trans-S/v/t
```



### 17.6.4 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 17.6.3 affect a block only if it is both active and visible. If no block is active, toggling with **g** just shows the wiring in the whole network.

### 17.6.5 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).

### 17.6.6 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 17.6.5).

Initially, just the wiring from the previous time-step appears, with the 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.

### 17.6.7 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 recursive inputs in section 17.6.6. 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 (section 17.6.1) 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 1, outputs=2/100, step-s all-def:** *(for a k=2, n=100 network)*



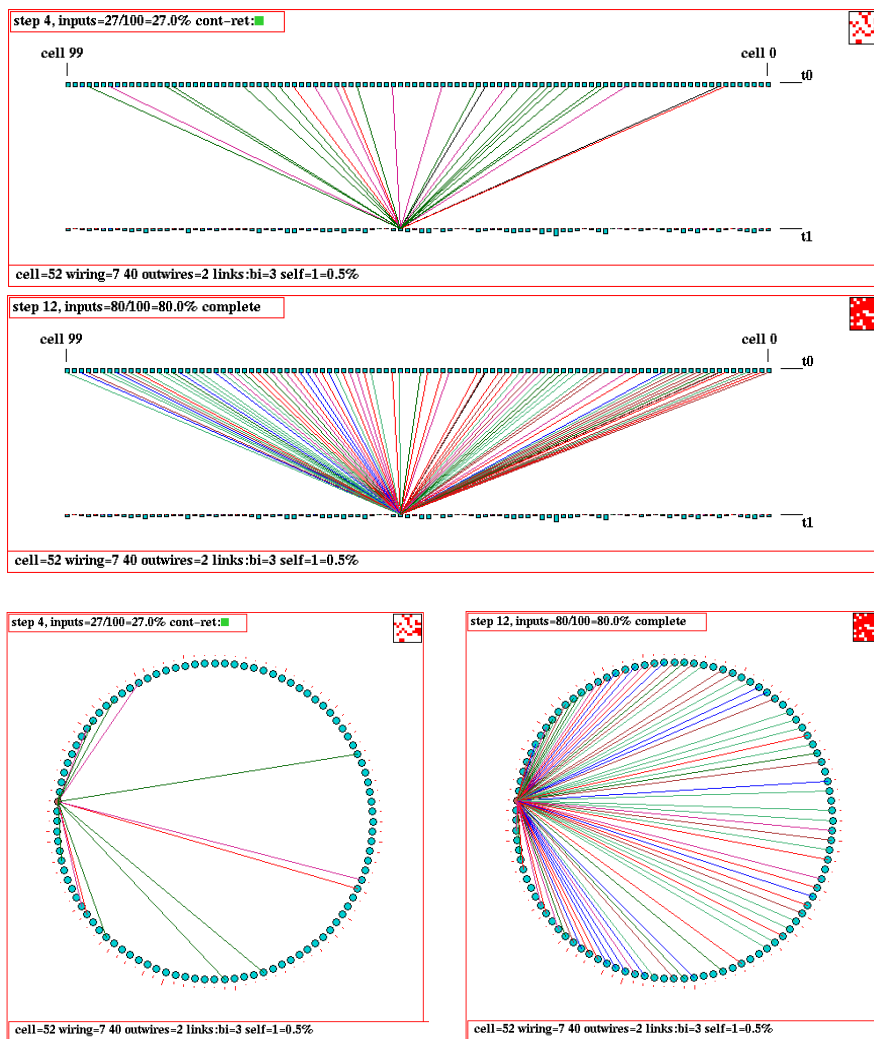


Figure 17.7: Recursive inputs (direct and indirect) to a given cell for a  $k=2$ ,  $n=100$  RBN. *Above:* As time-steps. *Below:* As a circle. Firstly, inputs are shown after 4 backward steps with 27 cells reached. Secondly, the complete inputs are shown after 12 backward steps with 80 cells reached. A small 2d insert in the top-right hand corner of the wiring window graphically indicates the fraction of cells connected so far.

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% cont-ret:** (*values shown are examples*)

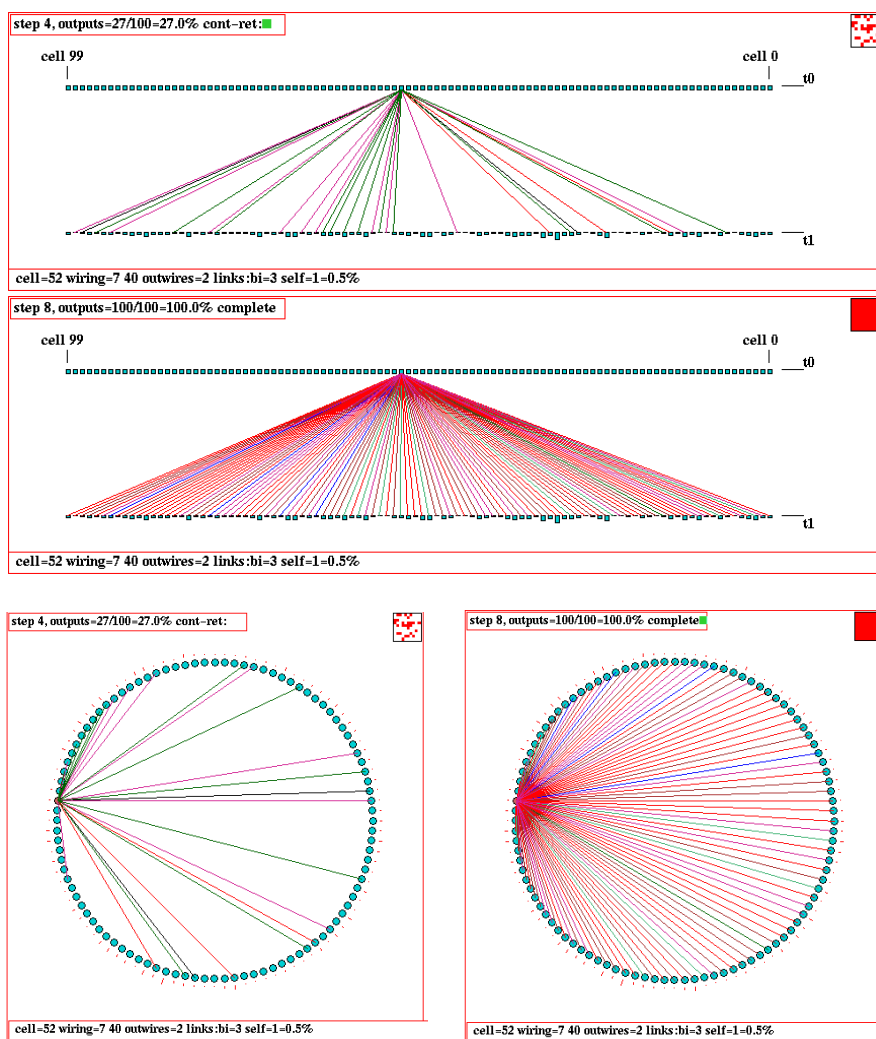


Figure 17.8: Recursive outputs (direct and indirect) from a given cell for a  $k=2$ ,  $n=100$  RBN. *Above:* As time-steps. *Below:* As a circle. Firstly, outputs are shown after 4 forward steps with 27 cells reached. Secondly, the complete outputs are shown after 8 forward steps with all 100 cells reached. A small 2d insert in the top-right hand corner of the wiring window graphically indicates the fraction of cells connected so far.

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.

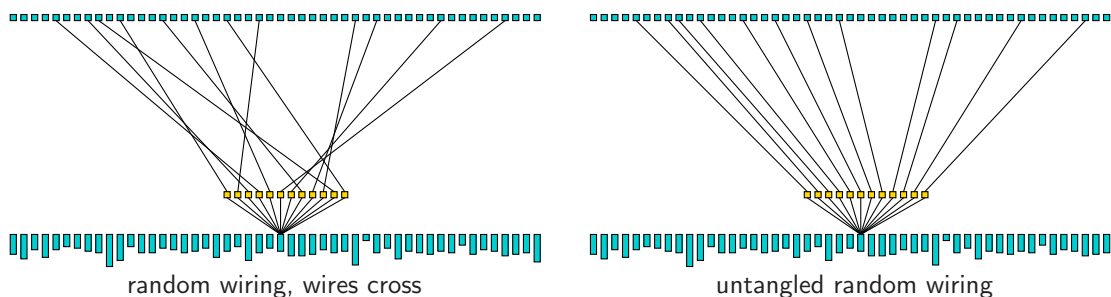


Figure 17.9: Untangling the wiring: wire connection points to the 1d pseudo-neighborhood for the active cell are rearranged so that the wires do not cross, and the rcode (if set as part of a rulemix) is automatically transformed to give equivalent behavior. In TFO-mode there is no need to transform the rule. In this example  $k=12$ ,  $n=50$ .

### 17.6.8 Untangling the wiring

Enter **u** in section 17.4 to “untangle” the wiring for the active cell, for the defined block if visible, or for 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 (for 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,

```
untangle 1d wiring: all-a, single cell-(def): (for the active cell)
or if a block is active and visible
untangle 1d wiring: block 11-22, all-a, single cell-(def): (for example)
```

If an rcode-mix was set, rules are transformed for equivalent dynamics<sup>2</sup>. For a single rcode network (section 14.1) the wiring will be untangled, but the rcode will remain unchanged — to get around 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.

### 17.6.9 Deleting a cell

*(1d networks, 1d or circle graphic, k-mix, rcode set, in main prompt sequence, not in TFO-mode)*

Enter **d** in section 17.4 to delete the active cell from the network and shorten the network by one cell, which automatically rewires the network and updates the 1d or circle graphic. The active cell must not be at index zero.

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 rcodes set (so not in TFO-mode). When a cell is deleted, all its 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.

<sup>2</sup>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.

## 17.7 Wiring graphic, 2d

Enter **2** in section 17.1, for the 2d graphic. This shows how a particular cell (the “active cell”), or a predefined 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 a mouse pointer/click or with the arrow keys, and its possible to “jump” to a new specified 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 17.7.4) as in figure 17.10. By default, “direct wiring” is displayed if the wiring is nonlocal, and the pseudo-neighborhood for local CA wiring. Enter **p** to toggle between the two. A 5-way toggle (enter **n**) alters the presentation of connections, and the cells connected, also useful in visualizing blocks. The active cell is colored red, its pseudo-neighborhood yellow, the “actual neighborhood” cells are colored green (figure 17.10).

Sections 17.7.1 — 17.7.8 below explain some of the options specifically for 2d in more detail that were summarized in section 17.15).

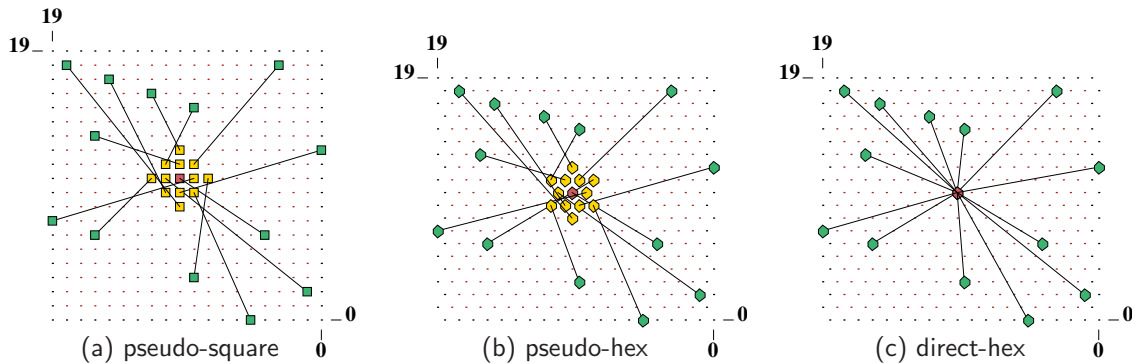


Figure 17.10: The 2d wiring graphic,  $n=20 \times 20$  RBN,  $k=13$ , with a central active cell. Toggle with *pseudo-p* between wiring to the pseudo-neighborhood — the default for local wiring, and direct wiring — the default for random wiring, as (b) $\leftrightarrow$ (c) above. Toggle with *toghex-x* between square and hexagonal layout and corresponding (default) neighbourhoods (section 10.1.3 and figure 10.2), as (a) $\leftrightarrow$ (b) above. The “active” cell is repositioned by mouse pointer clicks or moved with arrow keys.

### 17.7.1 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 17.9.1 to decode this data. The current cell, and cell input positions, are shown as  $I, J$  coordinates. The rule and rule data are shown in the rule window (section 16.19).

## 17.7.2 Moving or jumping between cells, 2d

To move the active cell, position the pointer and click any mouse button, or 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  $20 \times 20$ )

## 17.7.3 Alternative wiring presentation, 2d

A 5-way toggle allows alternative presentations of the active cell, or an active block. Enter *cell-links-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 17.8.4.

For a 2d block, the alternatives are shown in in figure 17.13.

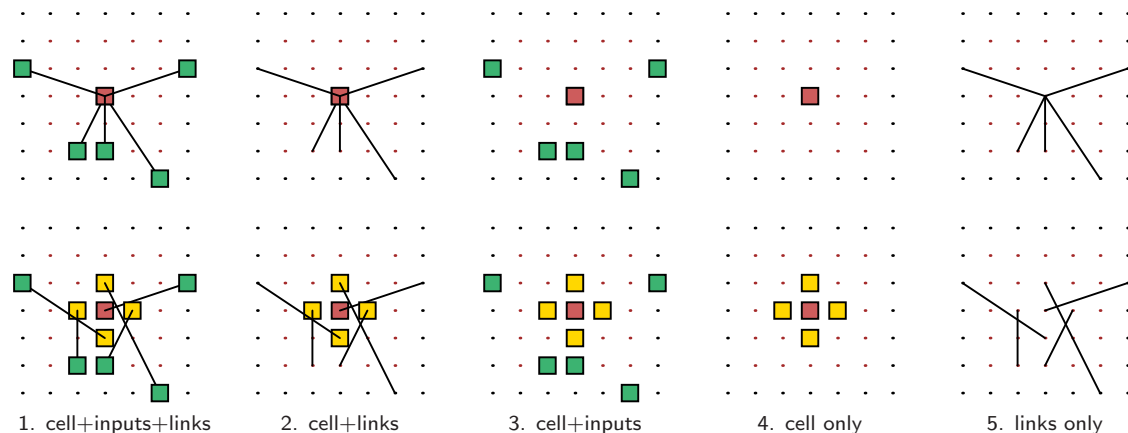


Figure 17.11: Toggling between 5 alternative ways of showing a cell, its inputs and connections, in 2d with *cell-links-n*, starting with the initial block presentation 1 (cell+inputs+links).

*Top Row:* shows direct wiring. *Bottom Row:* shows the pseudo-neighborhood toggled on (with *pseudo-p*, section 17.7.4).  $n=7 \times 7$  random (nonlocal) wiring,  $k=5$ .

## 17.7.4 Include the pseudo-neighborhood, or direct wiring only, 2d

By default, “direct wiring” is displayed if the wiring is nonlocal, and the pseudo-neighborhood is displayed for local CA wiring. Enter **p** to toggle between the two as in figure 17.10, and between the top and bottom rows of figure 17.11.

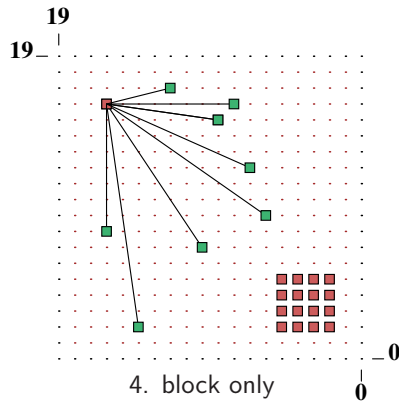


Figure 17.12: The 2d wiring graphic showing a block in full, i.e. a solid block as in the 1st edition of Exploring Discrete Dynamics. The new default presentation is to showing just the block edges as in figure 17.13, but this can be toggled to show the solid block with **G** (section 17.7.6). The active cell is shown here with direct wiring — without its pseudo-neighborhood (section 17.7.4).

### 17.7.5 Defining a block, 2d

Enter **b** to define a 2d block of cells. The following top-right prompt is presented<sup>3</sup>,

```
2d block-2, all-a, no block and single cell-def: (then, if 2 is selected ...)
2d block (max 19,19) (for a 20 x 20)
first corner (def 2,2) I:    J:
second corner (def 5,5) I:    J:
```

Enter **2** to define a block, **a** to define the whole network as a block. If **2** is entered, a subsequent prompt is presented for the first and second set of coordinates delimiting the block. The defaults correspond to the last two mouse clicks. Enter **return** to accept a default, or enter any required value.

As a consequence of larger network sizes allowing larger blocks, for 2d (and 3d) the new default presentation is to show just the block edges to speed up the graphics presentation (figures 17.13, 17.14), but this can be toggled to show the block in full (figure 17.12) as before, with *edges-G* (section 17.7.6).

As long as a block is active and visible (toggle with **g**), the 5-way toggle *block-links(x)-n* (section 17.7.3) applies to the block. The active cell will still be visible and can be moved, and its pseudo-neighborhood toggled with **p**, as usual.

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 — rcode-mix set)
```

... 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 (section 17.9.10).

```
... rule:Save/rev/trans-S/v/t
```

<sup>3</sup>For a 2d network shown in 1d, the prompt is as shown in section 17.6.3

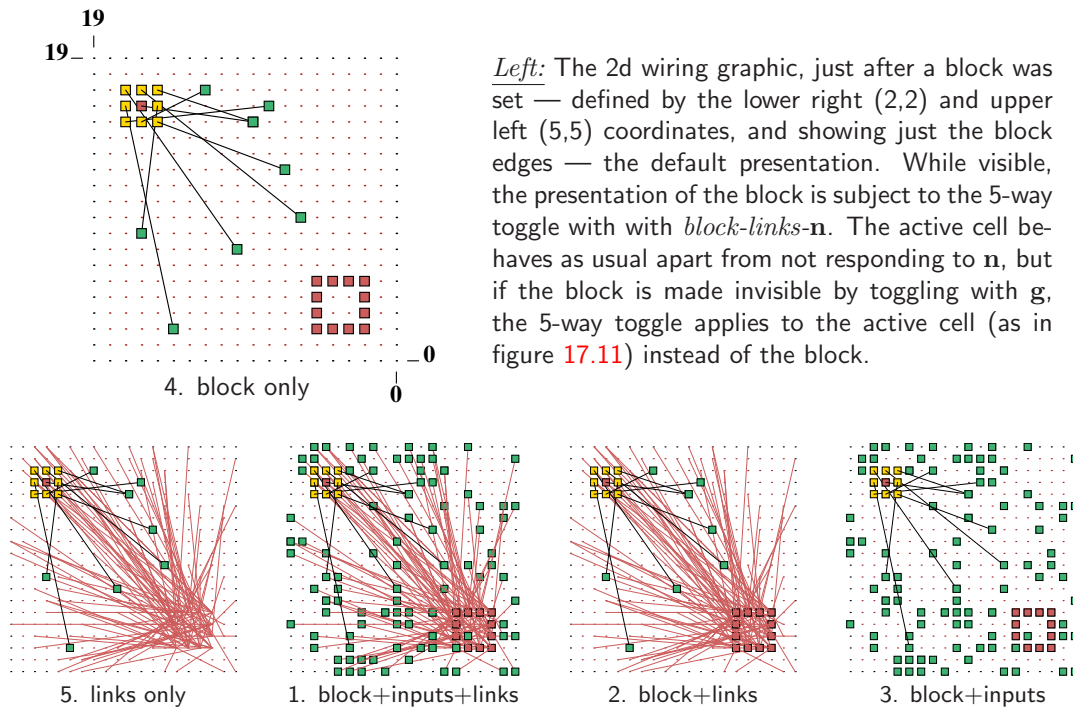


Figure 17.13: Toggling between 5 alternative presentations of a 2d block with `block-links(x)-n`, starting with the initial block presentation-4 (block only).  $n=20 \times 20$  RBN,  $k=9$ . The active cell is still visible and can be moved, and its pseudo-neighborhood toggled with `p`, as usual.

### 17.7.6 Toggling the block, 2d

If a block is active (section 17.7.5), enter `block-g` to toggle the block, making it either visible or invisible, without deactivating it. Enter `edges-G` to toggle between showing just the block edges (the default — figure 17.13) and a full or solid block (figure 17.12).

### 17.7.7 Expand/Contract the scale, 2d

The initial scale of the 2d wiring graphic is set automatically. Enter `e` to expand, `c` to contract this default scale. The contraction step is usually one pixel, but continues (by  $\times 0.9$ ) until the total image height is reduced to about 10 pixels. This allows for large 2d networks to be examined as in figure 17.14, but the background grid will disappear with contraction less than 3 pixels.

### 17.7.8 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 17.7.7. Enter `u` or `d` to move the graphic up or down by  $1/5$  of its vertical dimension.

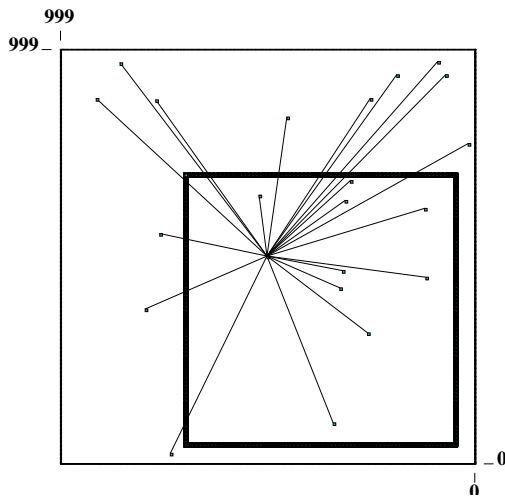


Figure 17.14: 2d wiring graphic, 1000x1000, showing a randomly connected central cell,  $k=21$ .

A “block+inputs” is active, its lower corner located at 50,50 and upper at 700,700, with just the block edges showing. Because the scale is less than 3 pixels, the background grid of dots does not appear. A randomly wired cell is shown at 500,500.

## 17.8 Wiring graphic, 3d

Enter **3** in section 17.1, for the 2d+3d graphic, where the network is shown simultaneously in 2d, and a 3d isometric projection<sup>4</sup> seen from below, as if looking up into a cage (figure 17.15). As in 2d, this shows how the active cell, or a predefined 3d block of cells, is wired in the 3d network. The 2d view and the 3d isometric can be independently expanded and contracted.

The 2d view shows successive horizontal slices (levels), stacked above each other, with the levels indicated. For larger networks the 2d view can be shifted up and down to see the relevant part. Otherwise the 2d view behaves very much as the 2d graphic described in section 17.7.

The 2d view acts as a sort of canvas for manipulating active cell position and simultaneously update on the the 3d isometric. This is done with a mouse pointer/click, or with the arrow and square bracket keys — `[`, `]` — for moving up and down between levels. There is also a “jump” option to a new specified location. The rule for the active cell appears in the rule window described in section 16.19.

As in the 2d wiring graphic, the display can be toggled between “direct wiring” and wiring to the pseudo-neighborhood. By default direct wiring is displayed if the wiring is nonlocal, and to the pseudo-neighborhood for local CA type wiring. Enter **p** to toggle between the two. A 5-way toggle (enter **n** alters the presentation of connections, and the cells connected, also useful in visualizing blocks. The active cell is colored red, its pseudo-neighborhood yellow, the “actual neighborhood” cells are colored green (figure 17.16).

Sections 17.8.1 — 17.8.9 below explain some of the options specifically for 3d in more detail that were summarized in section 17.4.1.

### 17.8.1 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 (figure 17.15). See section 17.9.1 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).

<sup>4</sup>Perhaps more accurately — a 45° oblique projection seen from within.



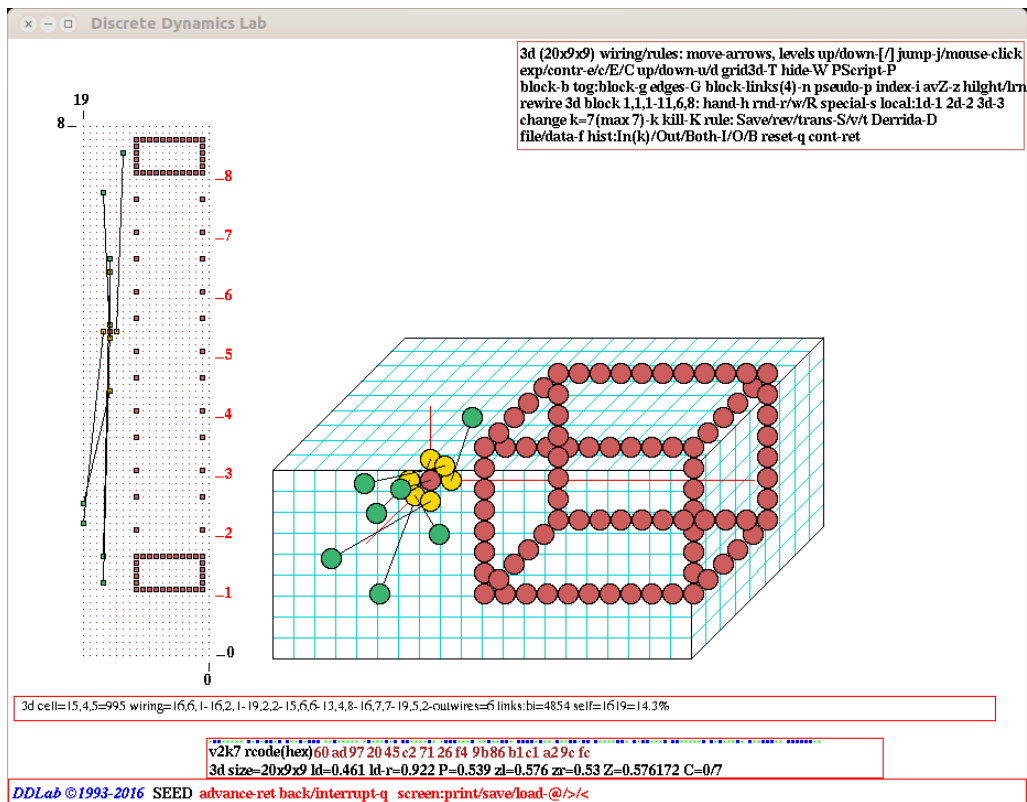


Figure 17.15: An example of the 3d wiring graphic ( $i, j, h=20 \times 9 \times 9$ , RBN,  $k=7$ ) as it appears on the DDLab screen. The active cell includes its pseudo-neighborhood, and a block (1,1,1 to 11,6,8) has been activated and its edges are visible. Rules have been set — the active cell's rule appears in the bottom panel. *Left*: the 2d view of the 3d network showing successive levels 0 to 9 stacked above each other (expand/contract with  $e/c$ ). *Right*: the network as a 3d isometric (expand/contract with  $\mathbf{E}/\mathbf{C}$ ). *Foot*: the wiring data. *Top Right*: the wiring graphic reminder.

## 17.8.2 Moving or jumping between cells, 3d

To reposition the active cell, click any mouse button with the mouse pointer on the 2d view. Alternatively use the arrow keys and square bracket keys —  $[, ]$  — for moving up and down between levels. There is also a “jump” option, enter  $j$  to jump to a specific cell 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):

Any change in the active cell on the 2d view will simultaneously update on the 3d isometric.

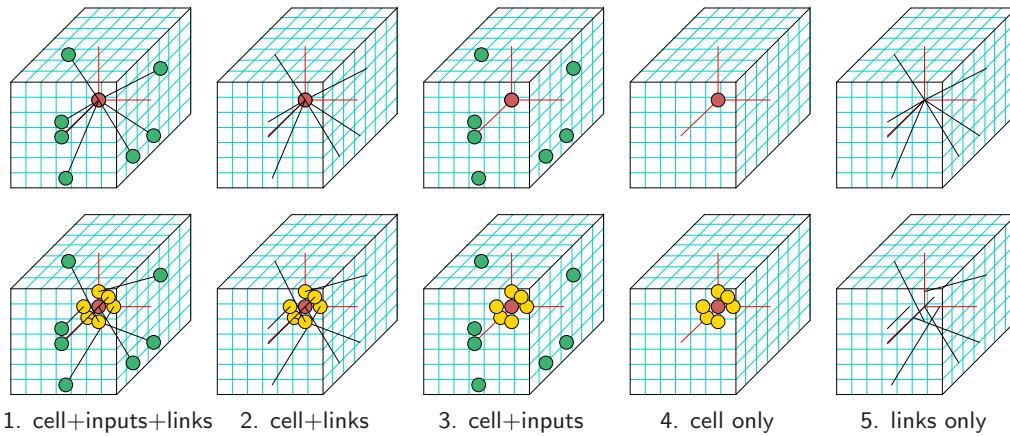


Figure 17.16: Toggling between 5 alternative ways of showing a cell and its connections in 3d with **n**. *Top Row*: with direct wiring. *Bottom Row*: with the pseudo-neighborhood toggled on (with **p**.  $n=7 \times 7 \times 7$  RBN,  $k=7$ ). These changes are seen simultaneously in the 2d view and the 3d isometric (figure 17.15).

### 17.8.3 Alternative wiring presentation, 3d

A 5-way toggle allows alternative presentations of the active cell, or an active block. Enter **n** to toggle between the five alternatives (figure 17.16) which appear simultaneously in the 2d view and the 3d isometric (figure 17.15). For the active cell, the 5-way toggle works both with direct wiring and with the pseudo-neighborhood (toggle with **p** section 17.8.4). The alternatives for a 3d block are shown in in figure 17.18.

### 17.8.4 Include the pseudo-neighborhood, or direct wiring only, 3d

As in 2d, by default, “direct wiring” is displayed if the wiring is nonlocal, and the pseudo-neighborhood for local CA wiring. Enter *pseudo-p* to toggle between the two, as between the top and bottom rows of figure 17.16.

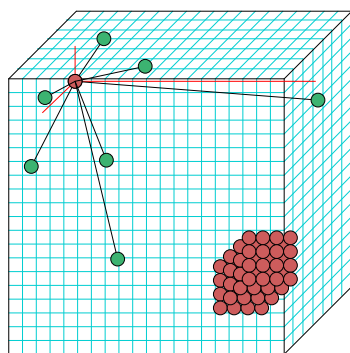
### 17.8.5 Defining a block, 3d

Enter **b** to define a 3d block of cells. The following top-right prompt is presented<sup>5</sup>,

```
3d range-3, all-a, no block and single cell-def: (then, if 3 is selected ...)
3d block (max 20,7,20) (for a 40 x 40)
first corner (def 2,2,2) I:   J:   H:
high corner (def 5,5,5) I:   J:   H:
```

Enter **3** to define a block, **a** to define the whole network as a block. If **3** is entered, a subsequent prompt is presented for the first and second set of coordinates delimiting the block. The defaults correspond to the last two mouse clicks. Enter **return** to accept a default, or enter any required value.

<sup>5</sup>For a 3d network shown in 1d, the prompt is as shown in section 17.6.3



4. (solid) block only

Figure 17.17: The 3d wiring graphic showing a block in full, i.e. a solid block as in the 1st edition of Exploring Discrete Dynamics. The new default is to showing just the block edges as in figure 17.18, but this can be toggled to show the solid block with **G** (section 17.8.6). The active cell is shown here with direct wiring — without its pseudo-neighborhood (section 17.8.4).

As a consequence of larger network sizes allowing larger blocks, for 3d (and 2d) the new default presentation is to show just the block edges to speed up the graphics presentation (figures 17.15, 17.18, 17.19), but this can be toggled to show the block in full (figure 17.17) as before, with *edges-G* (section 17.8.6).

As long as a block is active and visible (toggle with **g**), the 5-way toggle *block-links(x)-n* (section 17.8.3) applies to the block. The active cell will still be visible and can be moved, and its pseudo-neighborhood toggled with **p**, as usual.

An active block is indicated in the wiring graphic reminder (section 17.4),

**rewire 3d block 2,2,2-5,5,5: ...**

To deactivate the block enter **b** in section 17.4, then **return**. If the block is active and visible the following **rewire** options ...

**... hand-h rnd-r/w/R special-s local:1d-1 2d-2 3d-3**  
**change k=7(max 13)-k ...** (for example — *rcode-mix set*)

... 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 (section 17.9.10).

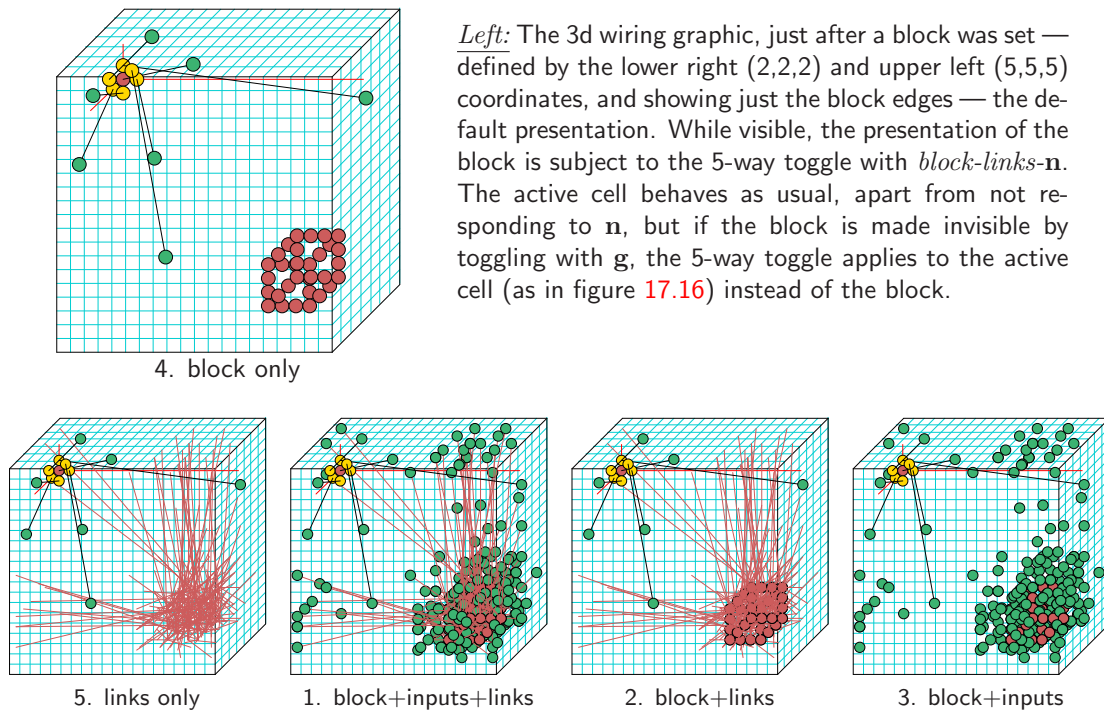
**... rule:Save/rev/trans-S/v/t**

### 17.8.6 Toggling the block, 3d

If a block is active (section 17.8.5), enter *block-g* to toggle the block, making it either visible or invisible, without deactivating it. Enter *edges-G* to toggle between showing just the block edges (the default — figure 17.18) and a full or solid block (figure 17.17).

### 17.8.7 Toggle 3d background grid

Enter *grid3d-T* to toggle the background grid on the isometric view, which is only visible when the scale is 3 or more pixels (expand/contract with **E/C**).



*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, and showing just the block edges — the default presentation. While visible, the presentation of the block is subject to the 5-way toggle with *block-links-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 17.16) instead of the block.

Figure 17.18: Toggling between 5 alternative presentations of a 3d block(with *block-links-n*, starting with the initial block presentation-4 (block only).  $n=20 \times 7 \times 20$  RBN,  $k=7$ . The active cell is still visible and can be moved, and its pseudo-neighborhood toggled with *p*, as usual. These changes are seen simultaneously in the 2d view and the 3d isometric (figure 17.15).

### 17.8.8 Expand/Contract the scale, 3d

The initial scales of the 2d+3d graphic are set set automatically, but can be independently expanded and contracted. Enter *e* to expand, *c* to contract the 2d view. Enter **E** to expand, **C** to contract the 3d view. The contraction step is initially one pixel, but continues (by  $\times 0.9$ ) until the total image height is reduced to about 10 pixels. This allows for large 2d networks to be examined as in figure 17.19, but the background grid will disappear with contraction less than 3 pixels.

### 17.8.9 Shifting the 3d graphic up and down

If only part of the 2d view 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 17.8.8.

Enter *u* or *d* to move the graphic up or down by about one level. Enter *s* to restore the default start position.

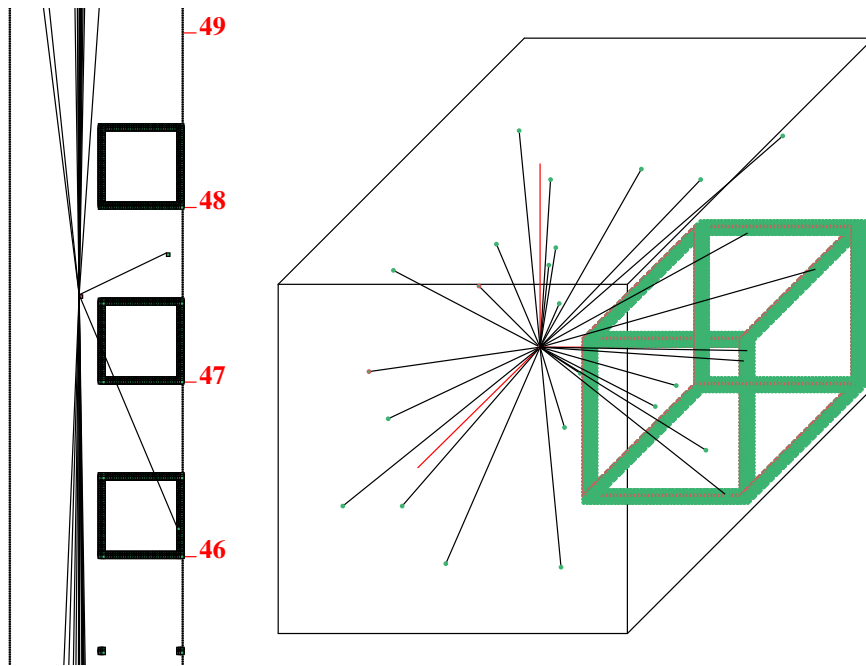


Figure 17.19: 3d network graphic 100x100x100, showing a randomly connected central cell,  $k=27$  ( $3 \times 3 \times 3$ ). A “block+inputs” is active, its lower corner at 2,2,2 and upper at 47,47,47, with just the block edges showing. Because the both the scale is less than 3 pixels, the background grids does not appear. A randomly wired cell is shown at 50,50. *Left*: the 2d view shifted down with  $d$  (section 17.8.9) to expose levels 46 to 49 revealing the top of the block. *Right*: the network as a 3d isometric.

## 17.9 Further options for the 1d, 2d and 3d wiring graphics

The following options which were marked with an asterisk in section 17.4.1 apply jointly to 1d, 2d and 3d networks, and are described in the rest of this chapter, sections 17.9.1 to 17.9.14 below,

### 17.9.1 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 active 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** ... the inputs — network indexes (for 1d) or coordinates (for 2d or 3d) of cells wired into the pseudo-neighborhood, ordered  $k - 1 \dots 0$ .

**outwires** ... the number of output wires, how many wires from the network are “plugged into” the active cell.

**self** ... the number of self-links, or inputs that are also outputs, from the active cell.

**bi-links** ... the number of cell pairs that have both inputs and outputs to each other.

The active cell’s rule, together with other details, (updated as the active cell is changed) are displayed in the lower rule window at the foot the DDLab screen (section 16.19).

## 17.9.2 Computing the (weighted) average $\lambda$ 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 by default for a  $k$ -mix. Enter **z** in section 17.4 to calculate and display the average, and weighted average,  $\lambda_r$  and  $Z$  parameters of the rules making up the rulemix. The data is displayed in a top-right window, for example, for a 1d network  $n = 150$ ,  $k$ -mix 3 to 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 local wiring and no rulemix the weighted average equals the average.

## 17.9.3 Options for learning pre-images

*rcode only — not in TFO-mode.*

Enter **L** in section 17.4 to start the “Learning, forgetting, and highlighting” functions (chapter 34). Learning/forgetting involves attaching/detaching sets of states as pre-images of a target state. For attractor basins, enter **I** to just highlighting selected states.

Learning/forgetting usually requires both nonlocal wiring and a rulemix. Algorithms automatically adapt the network’s wiring and/or rules to achieve the required transitions between states. The results and side affects of learning can be seen most clearly in a basin of attraction field, but the methods (allowing larger networks) also apply to a single basin, a subtree, or simply to a space-time pattern running forward. For further details refer to chapter 34.

## 17.9.4 Hand rewiring

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 17.20), including the following reminder, for example,

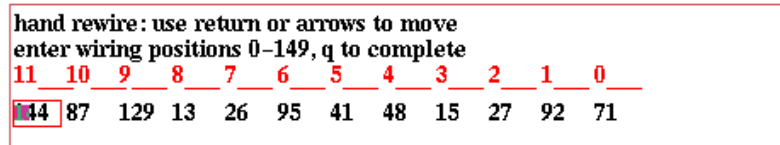


Figure 17.20: Hand wiring a single cell from the 1d wiring graphic,  $n=150$ ,  $k=13$

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 10.2.2 and 10.2.3.

### 17.9.5 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. Enter **w** to 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 local CA wiring) set in section 17.9.6 (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**.

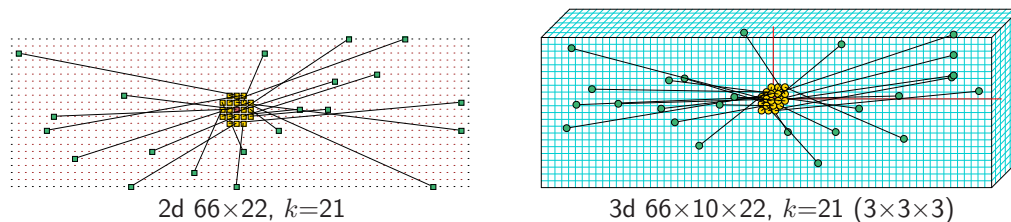


Figure 17.21: For homogeneous- $k$ , enter *rnd-R* for totally unbiased random wiring, ignoring any biases set in section 17.9.6 and with a reach that is not limited by the shortest axis. The examples show wiring to the pseudo-neighborhood.

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 9.7.2, entering **R** will rewire to give an approximate power-law distribution of outputs (figure 17.22), creating a “scale free” network, said to be characteristic of many natural and artificial networks, from metabolic networks to the world-wide-web [4]. The graph of such a network is shown in figure 20.2.

### 17.9.6 Biased random rewiring

If *special-s* 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 17.9.5 above. These biases apply to the active cell, or to the block (inside or outside) 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.

### 17.9.7 Local 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 local 1d, 2d or 3d CA-like wiring to the local neighborhood, with periodic boundary conditions. For 3d networks, 1d, 2d or 3d local CA wiring applies. For 2d networks, 1d, 2d local wiring CA applies. For 1d networks, only 1d local CA wiring applies. Note that CA wiring can also be set as “random wiring” in section 17.9.6 above, allowing various biases to the CA wiring as described in section 12.5.

### 17.9.8 Changing the neighborhood size, $k$

*r*code-mix set only — not in TFO-mode

Given a rcode-mix (section 14.1), and irrespective of  $k$ -mix, enter **k** in section 17.4 to change the neighborhood size  $k$  of the active cell, or of the block if active and visible. The following additional prompt appears,

**reset nhood size k (1-13):** (if  $k_{Lim}=13$ )

The maximum  $k$  allowed is  $k_{Lim}$  (table 7.1), which may be greater than the actual max- $k$  in the network (section 9.10). 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 local 1d, 2d or 3d (depending on the network dimensions) with periodic boundary conditions.

For a homogeneous- $k$  network, an individual cell or block 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 if  $k$  was reduced) is preserved. Any excess rule-table entries are set to 0.

Note that  $k$  may also be increased from the transformation options (sections 17.9.11, 18.7.1) giving a neutral transformation with equivalent dynamics.

### 17.9.9 Kill a cell

*r*code-mix set only — not in TFO-mode

Enter **K** in section 17.4 to kill or neutralize the active cell<sup>6</sup>, 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  $k=0$  (section 9.2). For  $v=2$  the decimal rcode is 2. The cell then 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.

---

<sup>6</sup>For 1d networks, a cell can also be deleted entirely (section 17.6.9).



## 17.9.10 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 rulemix, any revision applies to the whole network. Note that a rulemix 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 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 example,

**Copy rule to 33-55 -c:** (1d)

**Copy rule to 2d range 0,22-39,33 (880-1359)-c:** (2d, 40×40)

**Copy rule to 3d range 0,0,8-19,19,12 (3200-5159)-c:** (3d, 20×20×20)

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 (section 16.19).

To change a rule to one with a different neighborhood size  $k$ , first change  $k$  in the rewiring window (section 17.9.8), then change the rule.

## 17.9.11 Transforming the rule

*rcode set only — more options rcode-mix — more options k-mix — 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 in chapter 18. This option is intended for networks with rcode-mix or  $k$ -mix. For networks with homogeneous rcode 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 complemented, 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 transformations 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 (section 18.7.4) and automatically generating the minimal mixed- $k$  network that satisfies the mapping, (i.e. reduced to effective- $k$ ), one solution to the “inverse problem”.

## 17.9.12 Filing, from the wiring graphic

Enter **f** in section 17.4 for the network filing options where its possible to save, and load networks that fit into the “base”, thus to combine networks and create networks of sub-networks, described in chapter 19. To load a network file, first set up a compatible or “dummy” network (section 19.4). The complete network architecture, or just its wiring or rulemix, can be saved, and loaded into a base network. The options vary according to the  $k$ -mix, rules set, and other constraints.

When a file is loaded, it is automatically treated as a block and located by a prompt, 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 becomes active with its edges apparent in the wiring graphic, framing the whole graphic for a file/base of equal size.

Network data can also be printed to a file, to the xterm window for Linux-like systems, or to a printer for DOS (section 19.6).

### 17.9.13 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 9.11. Examples of power-law distributions are shown in figures 17.22. 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,

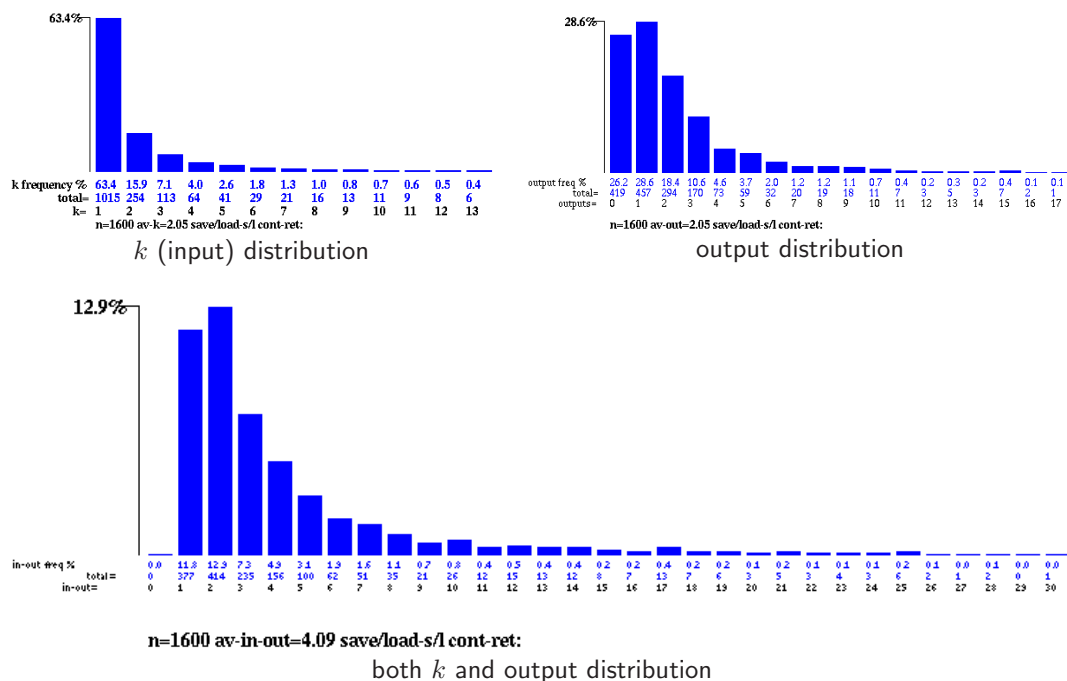


Figure 17.22: Histograms of a power-law distribution of network links, ( $k$ ) inputs, outputs, and both combined. The power-law exponent was set to 2.0 in section 9.7.2 for a mixed= $k$  network 40×40. Random wiring was reset for the whole network with **R** (section 17.9.5), where outputs are preferentially allocated according to  $k$ .

**n=100 av-k=2.02 save/load-s/l cont-ret:** (*for example*)

**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** to save or load the histogram data as a **.his** file (section 35.3). In both cases the data appear in the xterm window (*not for DOS*), for example for figure 17.22 *Top Left*,

```

histogram: 1+13 columns
0 1015 254 113 64 41 29 21 16 13 11 9 8 6

```

### 17.9.14 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 recording exactly what appears 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, 17.19, 17.21, and others in this chapter, which were cropped with the methods described in section 36.1.

---

# Chapter 18

## Transforming rules

*not in 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 17.9.11
- space-time pattern interrupt options, section 32.16.1.
- 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 rulemix, transformations apply to the whole network. For a rulemix from the wiring graphic, transformations apply to the active cell or a predefined block. When interrupting space-time pattern or attractor basin, the required cell index is first selected with an upper-right prompt, similar to the examples below,

**enter cell index, 1599-0:** (*this example for 2d space-time patterns<sup>1</sup>, 40×40*)

**enter cell index, 11-0:** (*this example for an attractor basin, n=12*)

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 transformation options include,

- inverting the rcode between two conventions for listing neighborhoods (section 18.3).
- various manipulations of the rcode table, including “solidifying” (section 18.4) and negative and reflection transformations within equivalence classes and rule clusters [31] (section 18.5).
- setting or amending canalizing inputs ( $v=2$  only) (section 18.6).
- transforming the neighborhood<sup>2</sup>, equivalent rcode with greater  $k$ , or with smaller “effective- $k$ ” (section 18.7).
- creating a network to satisfy an exhaustive mapping of transitions ( $v=2$  only) (section 18.7.4).
- Showing Equivalence Classes or Rule Clusters in the terminal (section 18.5).

---

<sup>1</sup>Note that the cell index  $x$  is expressed as 1d, even if the network is 2d or 3d — sections 10.2.2 and 10.2.3 explain how to convert  $x$  to 2d or 3d coordinates, and vice versa.

<sup>2</sup>Transforming the neighborhood does not apply when interrupting space-time patterns (section 32.16.1) or attractor basins.

## 18.1 Options for transforming rules

Examples of the transformation options are show below. They vary according to the context<sup>3</sup> depending on factors such as the rulemix,  $k$ -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: invert-v solid-o comp-m neg-n ref-r canal-C
equiv>k(3):(4-13), eff k(if k=3)
save/prtx: rcode-s/x EquivClass-E RuleCluster-R:
or if totalistic, kcode (or tcode)
save/prtx: rcode/kcode-s/S/x/X EquivClass-E RuleCluster-R:
```

### 18.1.2 Transform options, mixed- $k$

For a mixed- $k$  network the transformation options are invoked from the wiring graphic in section 17.9.11. Then the first two lines of the prompt are ...

```
transform rule: invert-v solid-o 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)
```

If  $k=k_{max}$  `equiv>k...` is omitted.

For  $v=2$  only, and  $k_{max} = n$  and  $n \leq 13$ , there is an additional option to load an exhaustive map (`.exh`) file (see also 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, homogeneous- $k$

For a mixed rule network with homogeneous- $k$  network, the “greater  $k$ ” and “effective  $k$ ” transformations do not apply.

```
transform rule: invert-v solid-o comp-m neg-n ref-r canal-C (all+a)
save/prtx: rcode-s/x EquivClass-E RuleCluster-R:
```

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.

---

<sup>3</sup>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

For a single rule network, transformations described in sections 18.3 to 18.6 apply to all cells, whereas for a mixed rule or mixed- $k$  network, just the selected cell in the wiring graphic (section 17.9.11) can be transformed, or all cells in the network.

To transform all cells, add **a** to the selection, i.e. enter **va** in section 18.3, **oa** in section 18.4, **ma** in section 18.5.1, **na** in section 18.5.2, **ra** in section 18.5.3 and **Ca** in section 18.6. Entering **Ca** allows canalizing inputs for the whole network to be reset as described in chapter 15.

Note that for a mixed rule or mixed- $k$  network, all rules in a predefined block can be transformed if invoking transformations from the wiring graphic (section 17.9.11).

## 18.2 Saving or printing the transformed rule

The transformed rule (rcode) is displayed in the rule window. Enter **s** at prompt in section 18.1.1 to save the rule (section 35.3), 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 — the kcode or tcode.

## 18.3 Inverting the rcode

The rcode-table (or any rule-table) can be expressed according to two alternative conventions, where the output of the all-0s neighborhood is either 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 (sections 13.3, 13.5), so that a rule-table string can be expressed as a decimal number following Wolfram convention [28] (if within the limits in section 16.6). Other conventions (e.g. [18]) follow the opposite order.

Enter **v** (or **va**) at the prompt in section 18.1 to invert the rcode/s between the two conventions. Effectively, the outputs of complementary 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  $v2k7$  “density classification” CA rcode [18] inverted to the convention in DDLab.

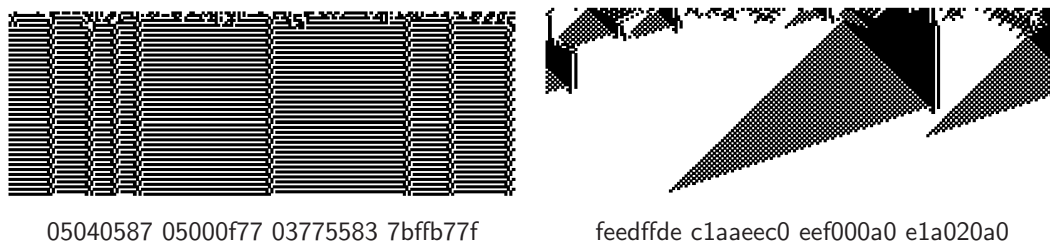


Figure 18.1: Transforming rcode: *Left*: the space-time patterns of a 1d density classification rule from [18] does not appear as intended because the order convention is the opposite of DDLab's.

*Right*: after the rule is inverted to DDLab's convention the density rule behaves correctly.  $n=199$ ,  $v2k7$ . The same random initial state is used in both cases. The rule-tables are shown in hexadecimal.

---

## 18.4 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 behavior. For binary ( $v=2$ ) “on” (i.e. “1”) cells will remain on. For  $v \geq 3$  a cell with the highest value ( $v-1$ ) will remain on.

---

## 18.5 Equivalence classes and rule clusters

As described in [31], a binary 1d rcode has equivalent rcodes by three symmetry transformations, negative, reflection, and negative+reflection in either order (figure 18.2), described in sections 18.5.2 and 18.5.3 below. The transformations make potentially four equivalent binary rcodes — there may be fewer because a transformation may lead to identity. For example, the rule-space of 256  $v2k3$  elementary rules [28] collapse into 88 equivalence classes, so only 88 prototype rules are required to fully describe rule-space behavior [31]. The rcodes are equivalent in the sense that a transformed initial state will make equivalent transformed space-time patterns, and attractor basins states are equivalent in the same way — their topology is identical.

Complementary transformations (section 18.5.1) will further group the rcodes into 48 rule clusters [31] which share a number or properties, including in-degree,  $g$ -density and  $Z$ -parameter.

For  $v \geq 3$  there will in principle be more equivalent rules in an equivalence class by permutations of values, but in DDLab at present the equivalence class size is still limited to 4 rules and the rule cluster to 8 rules because the complementary (thus negative) transformations are not computed exhaustively, but simplified as described in section 18.5.1.

Enter **E** or **R** to print the equivalence class or rule cluster in the terminal. Examples of rule clusters are shown below. Starting with a rule, n=negative, r=reflection, nr= negative+reflection, and complements are shown on the second line,

*for (elementary) rcode 110, in decimal*

110 n=137 r=124 nr=193

145 n=118 r=131 nr=62

*for tcode 35, k=5, in decimal — n= and nr=identity so not shown*

14 n=35

49 n=28

*for a v3k3 rcode, in hex*

160088268a45a9 n=10164204a22a85 r=1a24a4458a0289 nr=120a8216642481

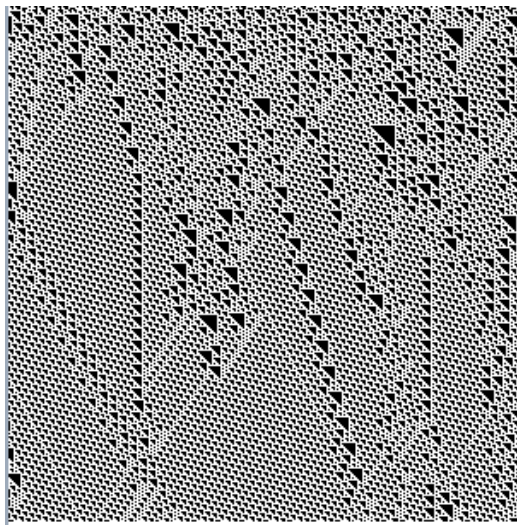
14aa2284206501 n=1a9468a6088025 r=1086066520a821 nr=18a02894468629

For binary rules within the decimal rule limits (section 16.6) rcode is shown in decimal if  $k \geq 3$ , and kcodes/tcodes if  $k \geq 5$ . Otherwise rules are shown in hex.

### 18.5.1 Complementary transformation

If **m** (or **ma**) is entered at the prompt in section 18.1, the rcode (or rcodes) will be transformed to its “complement”. For binary ( $v=2$ ) this is done by flipping all entries in the rcode-table, 1 to 0 and 0 to 1. For  $v \geq 3$  each entry  $a$  is changed to its simplified “complement”  $a_c = (v - 1) - a$ . See section 16.20 for a further explanation. A complementary rcode is usually not equivalent to the original.





rule 193

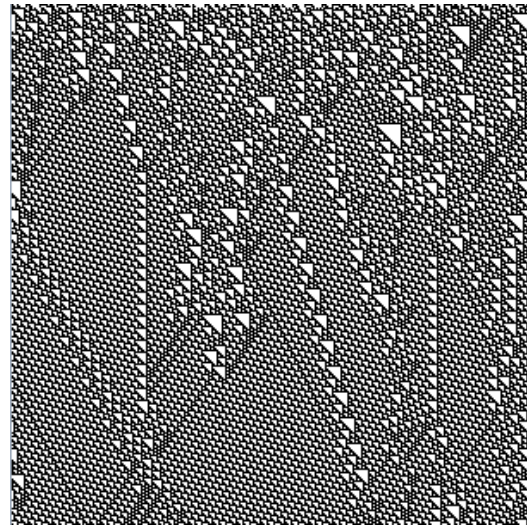
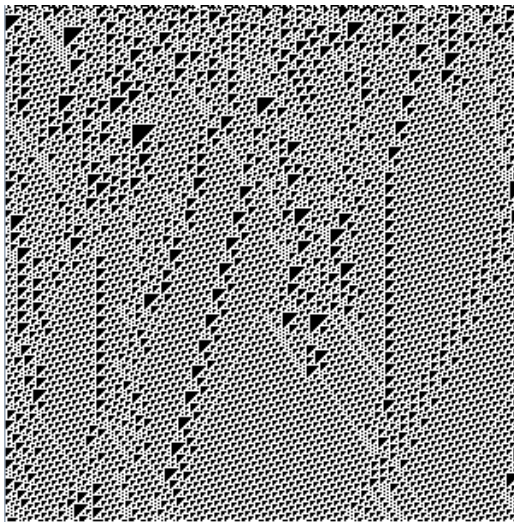
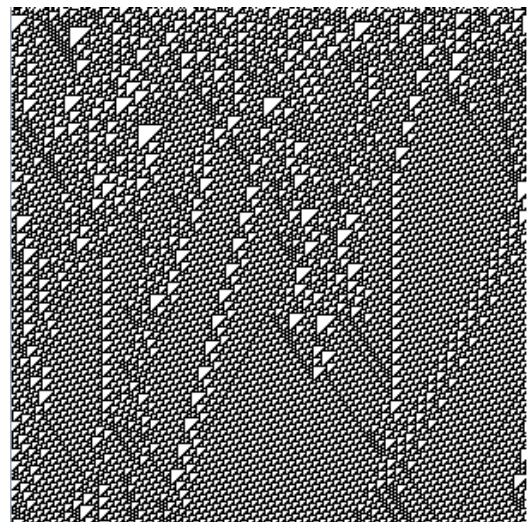
rule 124 ( $193_n$ )rule 137 ( $193_r$ )rule 110 ( $193_{nr}$ )

Figure 18.2: Equivalent transformations of the elementary rule  $v2k3$  rcode (dec)193, giving (negative) rule 124, (reflected) rule 137, and (negative+reflected) rule 110 (in either order). The initial state  $E$  has also been transformed to  $E_n$ ,  $E_r$ , and  $E_{nr}$  to produce equivalent space-time patterns — this is done in section 21.4.2, where *comp-m* gives the complement or negative transformation for a binary state, and *flip-h* (horizontal flip) gives the reflection.



### 18.5.2 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 complementary neighborhood pairs in the rule-table, then transforming the resultant rule-table into its complement. A negative rule and the start rule have equivalent dynamics, but with all state configurations in the space-time pattern complemented.

Note that a negative transformation followed by a reflection transformation (section 18.5.3), or vice versa (the order is equivalent) produces another equivalent rule, making 4 in all. For certain rules the transformations result in identity.

### 18.5.3 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 18.5.2, (or vice versa — the order is equivalent) produces another equivalent rule, making 4 in all [31] (figure 18.2). For certain rules the transformations result in identity.

## 18.6 Setting canalyzing inputs, single rcode

*see also chapter 15*

If **Ca** is entered at the prompt in section 18.1 for a mixed rcode network, canalyzing inputs can be randomly biased for the whole network as described in chapter 15

If just **C** is entered, canalyzing inputs may be amended for a single rcode, or for the active cell in a mixed rcode network. A given number of the rcode’s  $k$  input wires may be set to be canalyzing, either explicitly or at random. The following top-right prompt is presented.

**canalyzing inputs: explicitly-e random-n:**

### 18.6.1 Canalyzing inputs at random, single rcode

Enter **n** to set a given number of inputs as canalyzing at random (which can be reduced as well as increased). The following prompt appears, enter the required number,

**canalyzing inputs=0, set new number (0-5):** *(for  $k=5$ )*

Once set, the prompt in section 18.6 will reappear.

### 18.6.2 Canalyzing inputs explicitly, single rcode

Enter **e** to set the canalyzing inputs explicitly, the following prompts appear in sequence,

**wire (0-5):**    **input value (1,0):**    **output value(1,0):** *(for  $v2k5$ )*

Enter the neighborhood index (wire) to be set as canalyzing, and the input and output canalyzing values,  $v-1$  to 0 (1 or 0 for binary). Once set, the prompt in section 18.6 will reappear. Other inputs wires may be explicitly set as canalyzing, but the output must be the same as before to preserve previously set canalyzing inputs (chapter 15).

## 18.7 Neutral transformations of the neighborhood $k$

*not applicable when interrupting space-time patterns (section 32.16.1) or attractor basins*

Options for neutral transformations of the neighborhood  $k$  allow  $k$  to be increased, or decreased to find effective  $k$ , and rules are adjusted to conserve dynamics. These options apply from the main series of prompts at the start, not when interrupting space-time patterns or attractor basins.

### 18.7.1 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<sup>4</sup>. Increasing  $k$ , for example from  $k3$  to  $k5$ ,  $k6$ , or more, 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 increasing  $k$ , for example the  $k3$  rcodes 110 and 54 to  $k5$  (section 32.11.5, figures 32.26—32.28).

For networks with a homogeneous rule the transformation applies to the whole network. In mixed- $k$  or mixed rule networks, the active cell in the wiring graphic (section 17.9.11), or a predefined block, is transformed

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 increased by one. Thus the original connections remain in the enlarged pseudo-neighborhood. If more than one wire is added these steps are repeated. Rcodes are transformed so that the added wires are effectively redundant and play no role in the dynamics<sup>5</sup>. 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.

### 18.7.2 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 deliberately set in section 9.10, or a network may have been loaded with a smaller  $k_{max}$  than the base network (section 19.4.4). Enter **M** to reduce  $k_{max}$  to the maximum  $k$  found in the network.

<sup>4</sup>The algorithm is designed for 1d networks; for 2d and 3d the transformation is not neutral.

<sup>5</sup>Added wires can be neutrally reduced to effective  $k$  in section 18.7.3 to revert to the original wiring and rcode.

### 18.7.3 Effective $k$

The pattern of values 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 18.7.1 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 (section 35.3) 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  $k_{max}$  to be reduced to the maximum  $k$  found in the network (section 9.10).

### 18.7.4 Reverse engineering - loading an exhaustive map

For a network where  $k_{max} = n$  and  $n \leq 13$ , an exhaustive list of transitions, a random map, that was previously created (section 29.8) may be loaded to “reverse engineer” 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 specified, and where  $k_{max} = n$  (the network size), so  $n \leq 13$  because for rcode  $k_{max}=13$  in DDLab.

The resulting 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 (section 18.7.3) is implemented, followed by the option, if required, to reduce  $k_{max}$  to the actual maximum  $k$  found in the network (section 18.7.2).

1. Enter **e** at the prompt in section 18 to load the mapping file and transform the network to a fully wired network, which can be seen by pressing **return** and reverting to “Reviewing network architecture, wiring and rules” (section 17).
2. Enter **K** to reduce the wiring to effective  $k$  (section 18.7.3). 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 (section 18.7.2).

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 exhaustive map.

---

# Chapter 19

## File/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 17.6.3, 17.7.5 or 17.8.5), with its wiring visible in the wiring graphic, where the block can be separately 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 homogeneous-*k* (**k-hom**) and mixed-*k* (**k-mix**), and also depends on the context, where **n/a** signifies non-applicable — two examples are shown below,

*for homogeneous-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<sup>1</sup>*)

**wiring-only: print-pw, save-sw**

**rulemix-only: n/a** (*rulemix+wiring applies instead*)

**rulemix+wiring: print-pb, load-lb, save-sb:**

---

<sup>1</sup>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 isolation* may only be loaded earlier in the program sequence (section 9.4)

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** applies instead.

Subject to these constraints, to save, load or print the network architecture enter two key strokes as follows,

<i>first option</i>	<i>second option</i>
<b>p</b> - to print	<b>w</b> - for wiring
<b>l</b> - to load	<b>r</b> - for rules
<b>s</b> - to save	<b>b</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 filenames

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 extensions and default filenames, where  $x$  is 2 to 8, the value-range  $v$ .

	<i>extension</i>	<i>default filename</i>
<b>wiring-only</b> ...	<b>.w_s</b>	<b>my_wso.w_s</b>
<b>rulemix-only</b> ...	<b>.r_s</b>	<b>my_rso_x.r_s</b> for rcode
	<b>.r_v</b>	<b>my_rso_x.v_s</b> for kcode
	<b>.r_t</b>	<b>my_rso_x.t_s</b> for tcode
<b>rulemix+wiring</b> ...	<b>.wrs</b>	<b>my_wrs_x.wrs</b> for rcode
	<b>.wr_v</b>	<b>my_wrs_x.wr_v</b> for kcode
	<b>.wr_t</b>	<b>my_wrs_x.wr_t</b> for tcode

## 19.3 Wiring/rulemix encoding

The encoding<sup>2</sup> for all three file types above (section 19.2) starts with 6 bytes or 10 leading bytes depending on the network size  $n$ , 6 lead bytes for “small”  $n \leq 65534$ , or 10 lead bytes for “big”  $n \geq 65535$ . The leading bytes define  $v$ ,  $k$  or  $k_{max}$ ,  $n$ ,  $(i, j)$  the axes of a 2d network, and  $(i, j, h)$  the axes of a 3d network, where  $k_{max}$  = the maximum neighborhood size in a mixed- $k$  network

<sup>2</sup>See also the file encoding for single rules (section 16.16) and seeds (section 21.9).

(for homogeneous- $k$ ,  $k_{max} = k$ ),  $n$  is the network size. For a “small” network,  $n$  requires 2 bytes, and  $(i, j)$  requires 1 bytes each. For a “big” network,  $n$  requires 4 bytes, and  $(i, j)$  requires 2 bytes each. For 3d  $h$  is implicit in  $n$  and  $(i, j)$ . The encoding proceeds as follows<sup>3</sup>,

- 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 (“big”  $n$  byte 2,3,4,5) ... network size,  $n$ .
- byte 4,5 (“big”  $n$  byte 6,7 and 8,9) ...  $[i, j]$  which gives the dimensions and axis sizes.  
If  $i = 0$  and  $j = 0$  the seed is 1d. If both  $i > 0$  and  $j > 0$  the seed can be either 2d or 3d.  $h = \frac{n}{i \times j}$ . If  $h > 1$  the network is 3d, otherwise 2d.
- note axes limits ... for “small”  $n$  max axes  $(i, j, h) \leq 255$ , for “big”  $n$  max axes  $(i, j, h) \leq 65535$ . These limit are enforced when setting the axes in sections 11.6 and 12.3.

The encoding continues as follows, for each successive cell index from  $0 \dots (n - 1)$ , 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 but without the 2 leading bytes,

file type ... bytes required

**R: rulemix-only** ...  $R$  bytes, starting with the least significant bit/value in the rule-table.

**W: wiring-only** ...  $W$  bytes:

$W = 1 \times k_{max}$  if  $n < 255$ .

$W = 2 \times k_{max}$  if  $n \geq 255$  and  $n < 65535$  i.e. “small”.

$W = 4 \times k_{max}$  if  $n \geq 65535$  i.e. “big”.

This records the position of each input, indexed  $0 \dots k_{max} - 1$  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 rules and excess positions/bytes for wiring inputs 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.

<sup>3</sup>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.

**wiring** ... excess positions/bytes are filled with maximum values as follows:  
 with 255 (hex *ff*) if  $n < 255$ .  
 with 65535 (hex *ffff*) if  $n \geq 255$  and  $n < 65535$  i.e. “small”.  
 with 4294967295 (hex *ffffffff*) if  $n \geq 65535$  i.e. “big”.

Counting the number of positions/bytes not containing the maximum relevant maximum values (255, 65535, 4294967295) 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 9.2 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 completely 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 17.6.3, 17.7.5 or 17.8.5), with its wiring visible in the wiring graphic — the block can be separately 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.

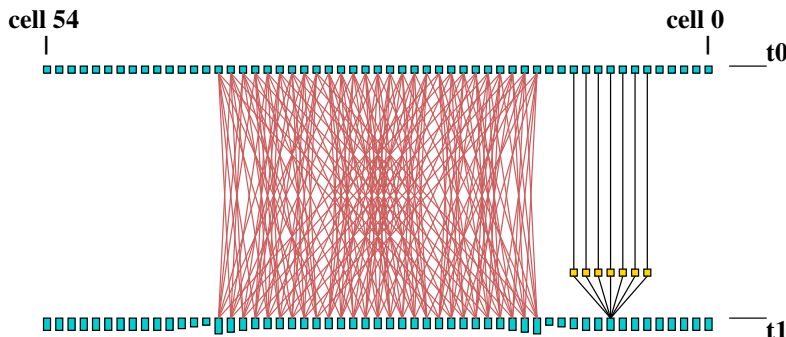


Figure 19.1: Loading a 3d wiring-only file ( $3 \times 3 \times 3$ ,  $k=6$ ) into 1d CA base network ( $n=55$ ,  $k=7$ ). The file appears in the 1d wiring graphic as a block (section 17.6.3).

### 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  $v$ , neighborhood  $k$ , 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 neighborhood  $k$ ,  $k_{max,file} \leq k_{max,base}$  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 many 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, it is 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  $i, j = 30 \times 1$  or 3d  $i, j, h = 30 \times 1 \times 1$ , 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,

**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.



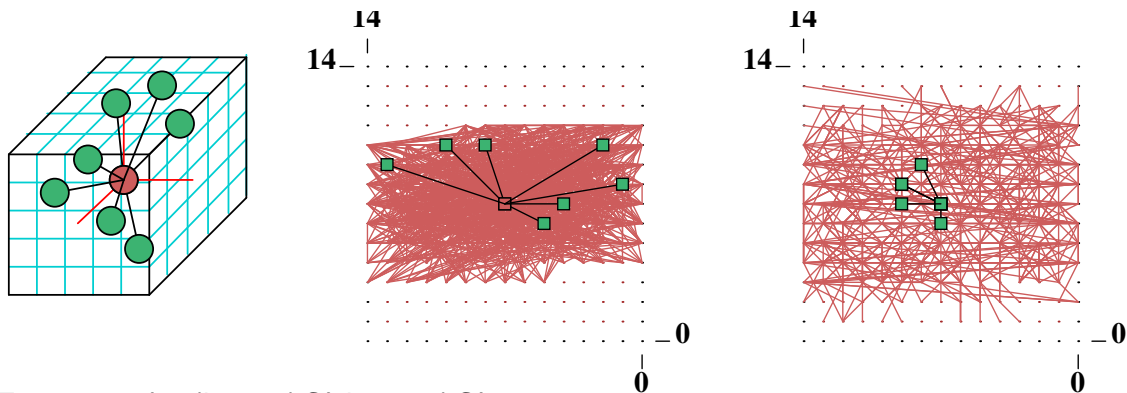


Figure 19.2: Loading a 3d CA into a 2d CA.

*Left:* a small 3d DDN ( $i, j, h = 5 \times 5 \times 5$ ,  $k=7$ ) was saved as a **rulemix+wiring** file. *Center:* the 3d DDN was loaded as a sub-network into the center 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 (1d) consecutive cells. When first loaded the sub-network is self-contained within its original boundaries — the dynamics within the sub-network would be independent of the rest of the network, but not vice 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.

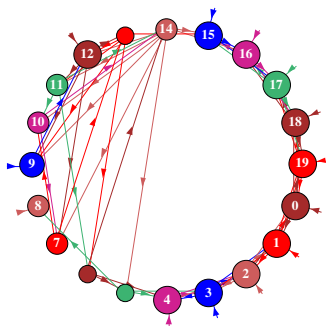


Figure 19.3: Loading a complicated sub-network ( $n=10$ ) with a rulemix,  $k$ -mix and nonlocal wiring into a simple 1d CA ( $n=20$ ). To do this the CA is set up to have a rulemix with just one rule. The figure shows the resulting network as a network-graph (chapter 20).

### 19.4.2 Loading a complicated network into a CA and vice versa

In general, to load a file network with a rulemix, the base also requires a rulemix, but this can be confined to a limited number of rules, or just one rule. What if a complicated file network with a rulemix,  $k$ -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 rulemix (section 14.4.3). Uniform  $k$  can also be treated as mixed- $k$  (section 9.9), and local wiring as nonlocal (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 have a rulemix with one rule.

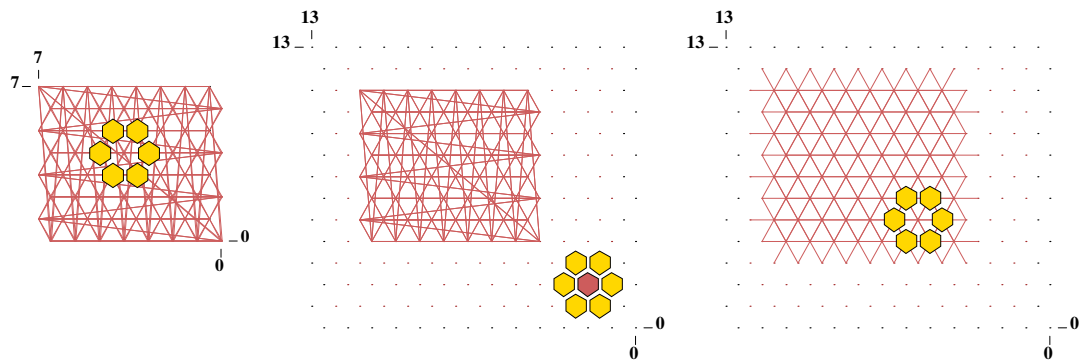


Figure 19.4: Loading a 2d CA into a larger 2d CA — showing a “solid” block “links only” (as in figure 17.12).

*Left:* a small 2d hex CA ( $i, j = 8 \times 8$ ,  $k=6$ ) was saved as a **rulemix+wiring** file.

*Center:* 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 vice 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.

### 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:**

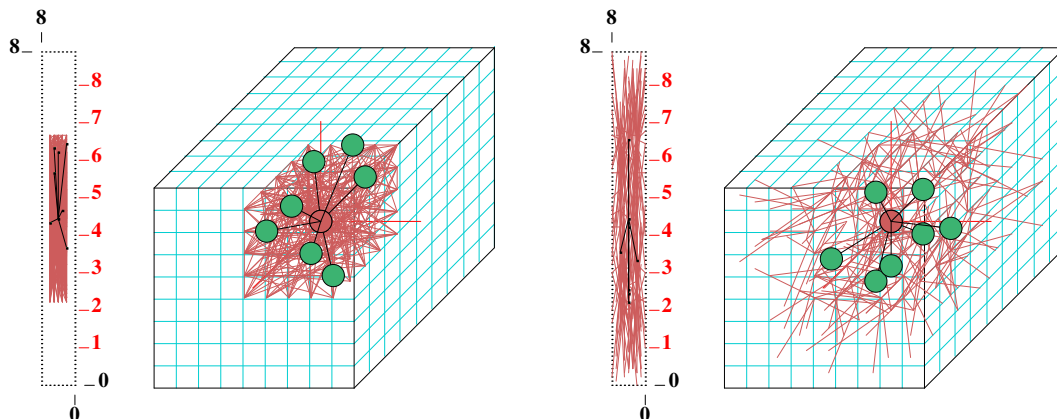


Figure 19.5: Loading a small 3d DDN into a 3d DDN — showing the 2d+3d wiring graphic with and block “edges” and “links only” (as in figure 17.18

*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 center of a larger 3d DDN ( $i, j, h = 9 \times 9 \times 9$ ,  $k=7$ ). 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 vice versa.

*Right:* The random wiring for the block was first confined to a 5 cell local zone (section 12.5.2), then its wiring was randomized by pressing **r** in the wiring graphic reminder, section 17.4. This connects the sub-network to its surroundings.

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 sub-network block will be visible and active in the base wiring graphic, where it can be be wired-in or otherwise manipulated, 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.

#### 19.4.4 Loading $k$ -mix networks

A  $k$ -mix network file can be loaded into a homogeneous- $k$  ( $k$ -hom) or a  $k$ -mix base network provided that the base max- $k$  is at least as big as the file max- $k$  (the precise base  $k$ -mix is not relevant). A base max- $k$  smaller than the file max- $k$  results in the following error message,

**file-kmax(5) > base-kmax(4), can't load, cont-ret:** (for example)

The max- $k$  in a  $k$ -mix network file 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 18.7.2.

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- $k$ .

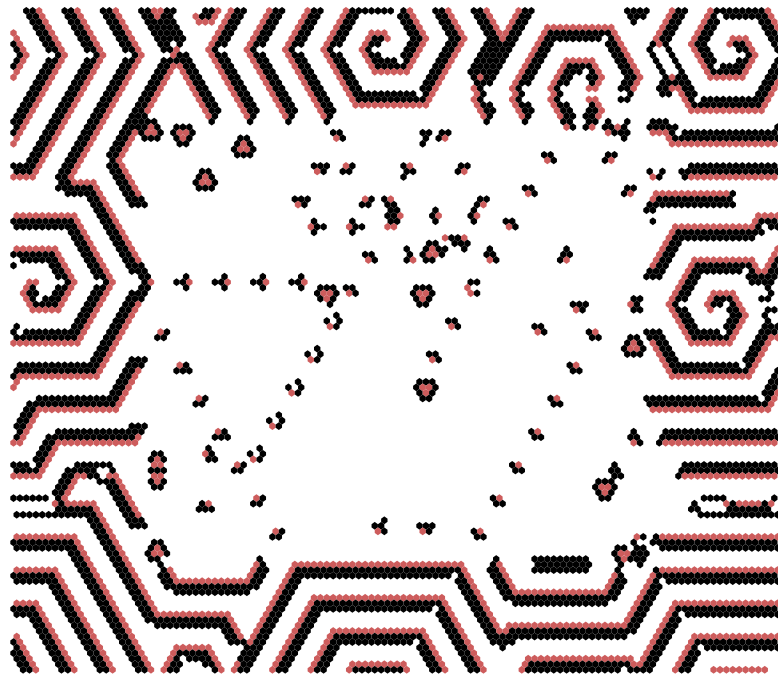


Figure 19.6: Space-time snapshot of a 2d CA inside another 2d CA with different  $k$ . The 2d ( $80 \times 80$ )  $k=7$  CA spiral-rule [45] ( $v3k7$  kcode(hex) 020609a2982a68aa64), was loaded into the 2d ( $120 \times 120$ )  $k=6$  complex CA which generates spiral structures ( $v3k6$  kcode(hex) 020282815a0254). See also figure 32.24.

---

## 19.5 Saving just the $k$ -mix

Enter **k** in section 19.1 to save just the  $k$ -mix (see also section 9.11.3). 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 9.4).

---

## 19.6 Printing network data to the terminal 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`: (`printer-p` for *DOS*)

Enter **p** to print the data to the terminal (xterm) 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` — section 35.3).

The way the data is presented depends on the selections **w**, **r** or **b** in section 19.1, mixed-*k* or homogeneous-*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 5x5
v2k4, 3d 6x6x6 n=276   ... for v=2, homogeneous-k, 3d network 6x6x6
```

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 totalistic rule was selected in section 13.1.1.

`ld ...`  $\lambda$  parameter.

`ld-r ...`  $\lambda$  ratio.

`Z ...` *Z* parameter.

`C ...` canalyzing shown in two parts (for *k*=3) firstly the fraction of canalyzing inputs (0/3, 1/3 *dotts*), then which of the *k* inputs are canalyzing, i.e. \*1\* — input 1 is canalyzing, 2\*\* — input 2, \*10 — both inputs 1 and 0. If there are no canalyzing inputs the second part is not shown.

`A ...` The Post function class, e.g. A[1]i (section 14.12). 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 (hex) rule-table itself can be very lengthy for larger *v*, *k* and *n*. However, the rule-table here is restricted to a maximum of about 80 hex characters<sup>4</sup> whatever the size of the rule<sup>5</sup>.

<sup>4</sup>This relates to the number of hex characters that would fit in the default rule window (section 16.19), which depends on the size of the DDLab screen itself (sections 6.3.1, 6.3.2), and also on the font size (section 6.4).

<sup>5</sup>There is no restriction on the size of the rule-table when printed to the terminal in section 16.18.

```

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 ffffffff 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 rule-tables (not in TFO-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

```

---

## 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 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 31.7.8, figure 31.20) where basin properties are found statistically by running a network forwards from many random initial states. As well as their standard presentation, space-time patterns can be run simultaneously within a network-graph allowing arbitrary layouts (section 32.19).

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 unravelling 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 Unravelling the jump-graph and the network-graph

The layout of the graphs can be rearranged 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

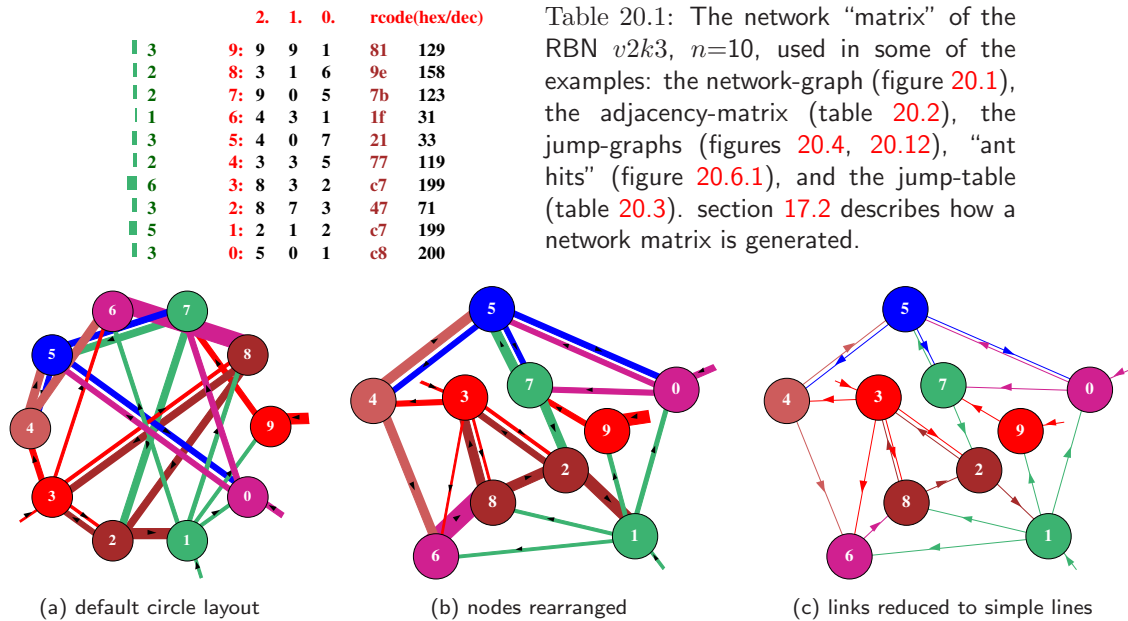


Figure 20.1: Network-graphs of the same RBN  $v2k3$ ,  $n=10$ , defined in table 20.1 showing alternative presentations (a) The default circle layout where edge width reflects the nodes’s outputs, and node diameter reflects  $k$ , the node’s inputs, all equal in this example. (b) Layout rearranged by dragging nodes with the mouse. (c) Directed links shown as thin lines with arrows.

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.

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.



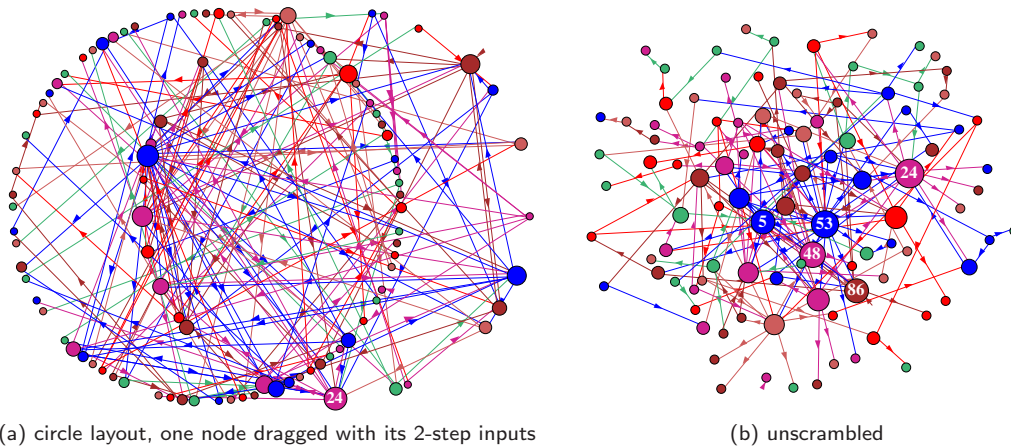


Figure 20.2: Two versions of the network-graph for the same network with a power-law wiring distribution, both inputs ( $k=1$  to 10) and outputs,  $n=100$ . To set up power-law wiring see sections 9.7.2 and 17.9.5. Nodes are scaled according to  $k$ . (a) A circle layout, but with one node dragged, together with its 2-step inputs. (b) the same network unscrambled — nodes rearranged as described in 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 32.16). 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 at 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 32.19), which can be in any graph layout. 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(tog)/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(tog)/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 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 summarized 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 that is 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 (section 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 position. For this, jump edges are unnecessary — if omitted (layout only) attractor nodes can be dragged more efficiently and CA compression can be retained.

As well as the complete basin of attraction field, the jump-graph can also be computed for the attractor histogram (section 31.7), 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 (section 1.6.1 for network size limits).

### 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 “basin 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,

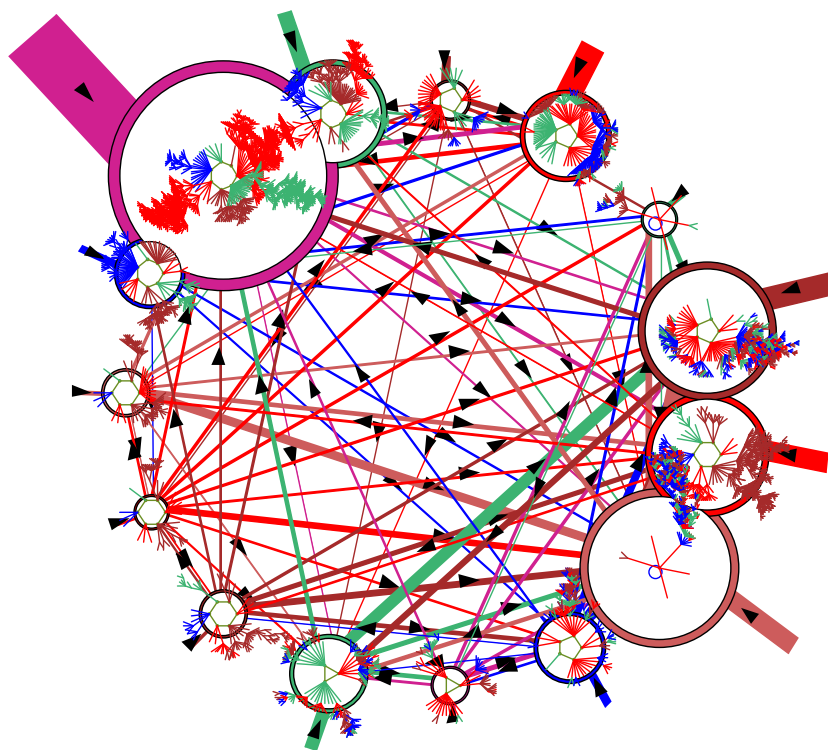


Figure 20.3: The jump-graph, with basins redrawn within its nodes (RBN  $v2k3$ ,  $n=13$ , as in figure. 2.5). The jump-graph shows the probability of jumping between basins due to single bit-flips to attractor states. Nodes representing basins are scaled according to the number of states in the basin (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 continue*)

### 20.3.2 The jump-graph reminder

If selected in section 20.3.1<sup>1</sup> (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,

<sup>1</sup>The jump-graph can also be selected in section 31.7.8 for the attractor histogram.

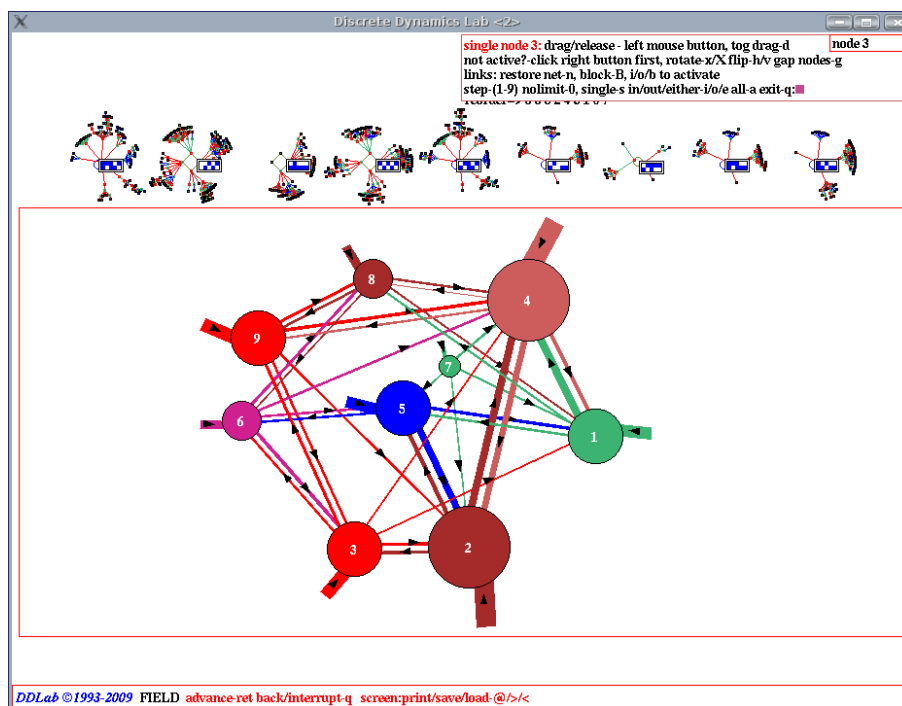


Figure 20.4: A screen shot of the basin of attraction field at the top of the screen and its jump-graph in a large window across the lower part of the screen. The jump-graph was adjusted and nodes were dragged to new positions with the left mouse button. Drag options are shown in the top-right “drag 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 toggled 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(tog)/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(tog)/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 44%,

**computing jump-graph, 25 basins, quit-q, or wait ...**

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.

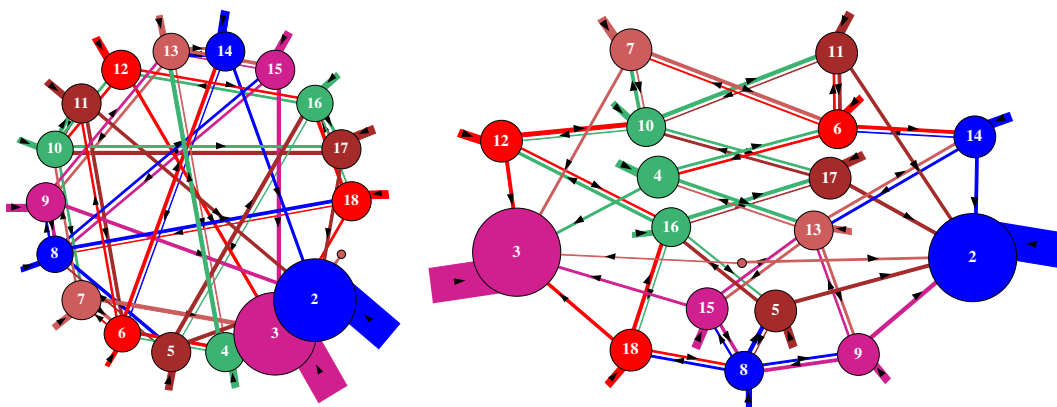


Figure 20.5: The jump graph for a 1d CA  $v2k3$ ,  $n=10$ , rule (dec)141. *Left*: the default circle layout, and *Right*: rearranged with the mouse to clarify the jumps, where the two largest basins (2 and 3) are completely stable.

## 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 (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 useful to uncover basins and basin details — a top-right inset will appear restore jump-graph -any key

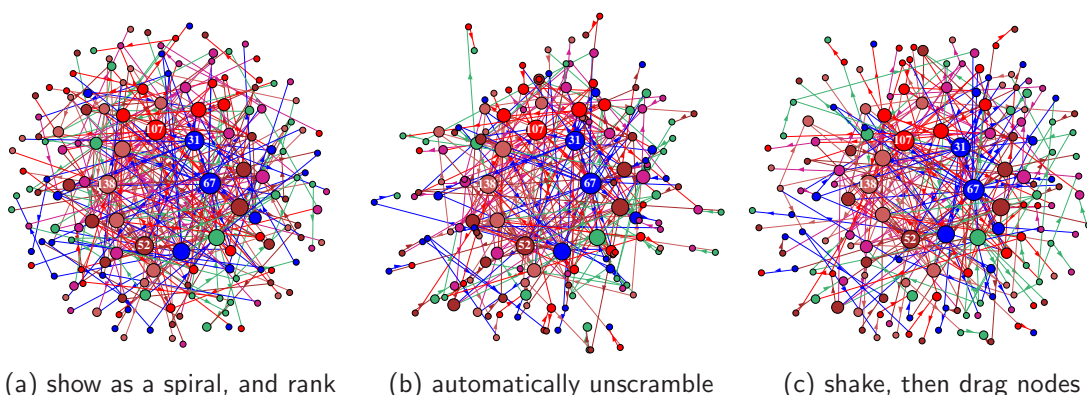


Figure 20.6: One possible method of unscrambling the network-graph of a scale-free RBN,  $n=150$ . (a): enter **O** to show the network as a spiral, then **k** to rank the nodes, placing nodes with the highest  $k$  near the center. (b): enter **u** to automatically unscramble the layout, possibly more than once. (c): 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 (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 (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, 20.12.
- Unreach-U** ... enter **U** to identify and separate unreachable or hard to reach nodes in the graph, (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 (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 (section 35.3).
- circle/spiral-o/O** ... enter **o** or **O** to show the graph as a circle or spiral. Circle layout is the default (except for the 2d or 3d network-graph). Spiral layout requires at least 30 nodes to be effective
- 1d/2d(tog)/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. The initial 2d network-graph is either square or hexagonal according to the underlying network — thereafter key *2d(tog)-2* toggles between square and hexagonal (examples in figures 10.5 - 10.7).
- 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-graph 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, 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.

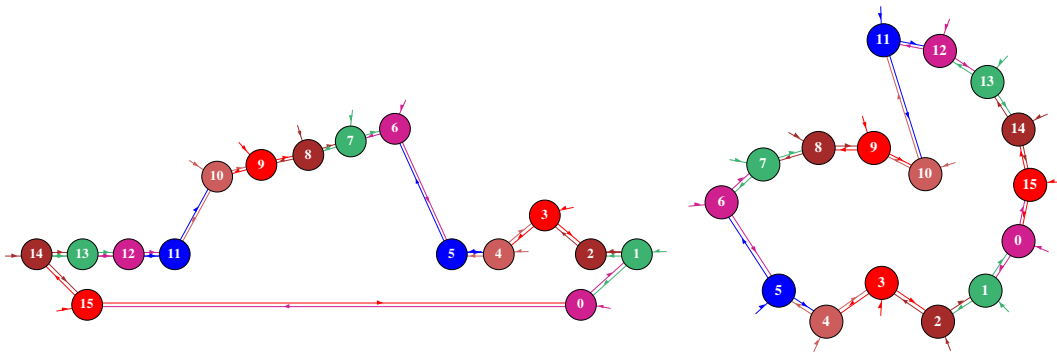


Figure 20.7: Dragging network-graph nodes and fragments of a 1d CA  $n=16$ ,  $k=3$ , starting with *Left*: a network-graph with 1d layout and *Right*: circle layout. Single nodes have been dragged, and in both cases node 8 and its 2-step neighbors have been dragged and rotated.

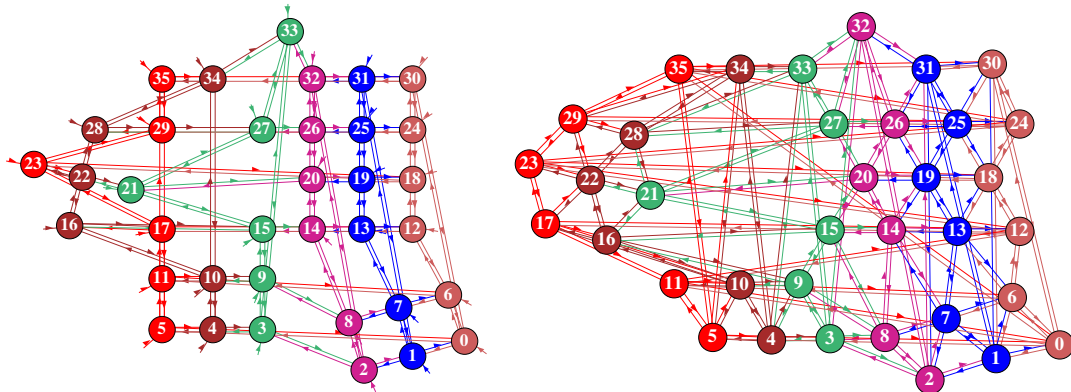


Figure 20.8: Dragging network-graph nodes and fragments starting with *Left*: a regular 2d network-graph of a 2d CA  $6 \times 6$ , square  $k=5$ , and *Right*: hexagonal  $k=6$  network. In both cases, a node has been dragged from the top row, and node 22 and its 1-step neighbors have been dragged and rotated, and a fragment (0 — 8) has been dragged 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,



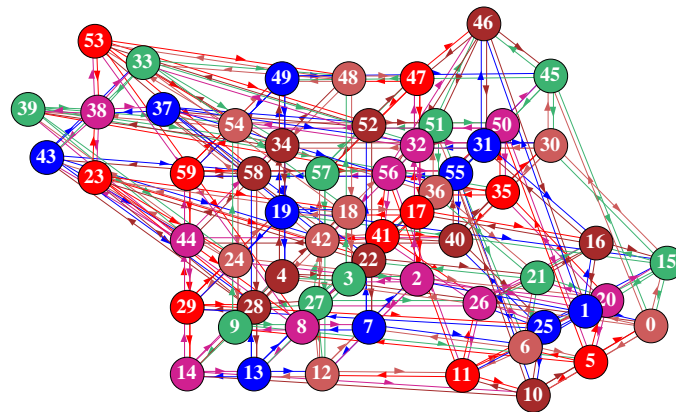


Figure 20.9: Dragging network-graph nodes and fragments starting with a regular 3d network-graph  $5 \times 3 \times 4$ ,  $k=6$ . Node 46 was dragged from the top level, node 38 and its 1-step neighbors were dragged and rotated, and a fragment (0 — 26) was dragged and rotated from the lower right hand 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, 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, 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 (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 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 button on a node to activate it and hold 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 pointer is 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 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 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<sup>2</sup> 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 17.6.3). The easiest way to set the block is to click (activate) the outer corners to set the defaults. Make sure these are really 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.

---

## 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<sup>3</sup> of ant hits. While active, a number of ant options are presented in a top right prompt as follows,

<sup>2</sup>The block methods are similar in chapter 17 “Reviewing network architecture” (sections 17.6.3, 17.7.5, 17.8.5).

<sup>3</sup>The maximum sample of ant hits is the maximum size of an unsigned long int -1 = 4294967295. If this is reached “Show ant hits” (section 20.6.1) is activated automatically.

*ant-prompt*

**probabilistic ant: quit-q speed- $\langle$ / $\rangle$  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 options in section 20.4.

**speed- $\langle$ / $\rangle$**  ... enter  $\langle$  to slow the ant down, or  $\rangle$  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** ... pause 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 eigenvectors 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.

### 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,

*ant 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:**

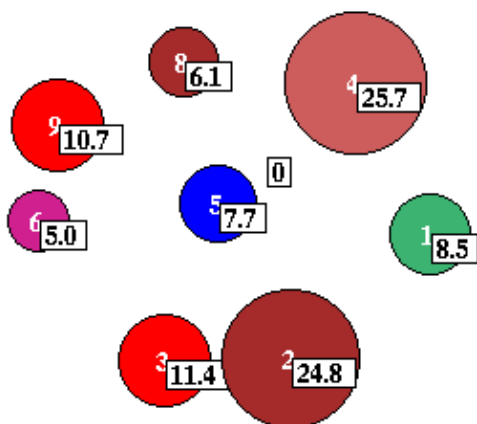


Figure 20.10: Probabilistic ant hits, or node visits (enter **h** followed **n** in section 20.6). This example is for the RBN  $v2k3$ ,  $n=10$ , defined in table 20.1. 60 million jumps took a few seconds. The percentage of hits/visits to each basin is shown, and can be compared with basin volume in figures 20.4, 20.12 and table 20.3. Node 7 is unreachable from other nodes so has zero hits.

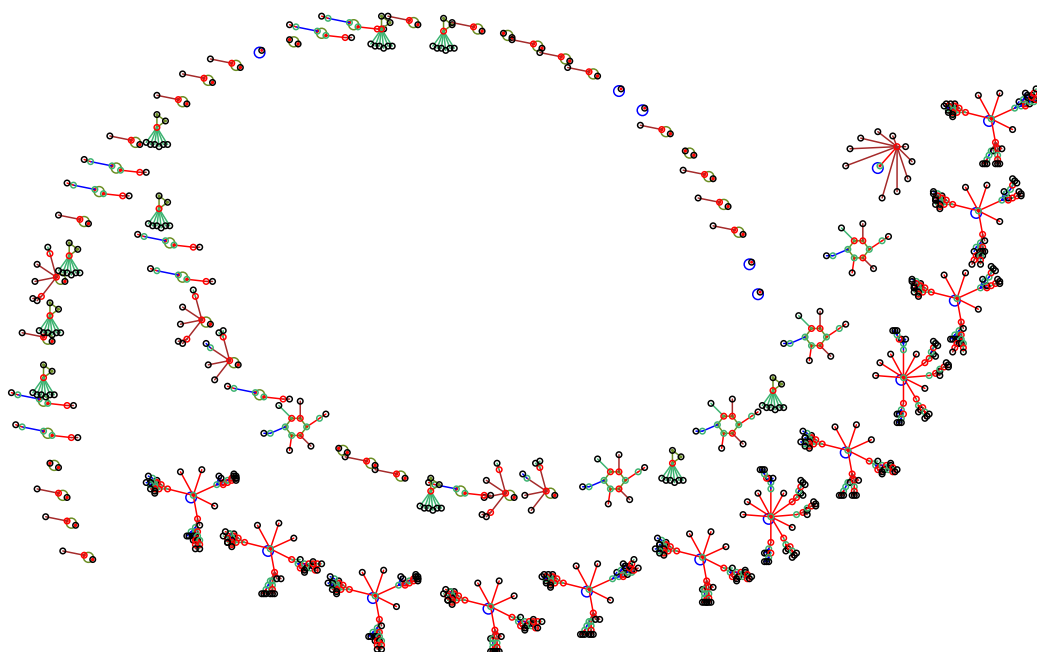
*options ... what they mean*

- 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 20.6.1).
- 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 can be somewhat difficult using the methods 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 26.2).

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 procedure. 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.



Above: all 73 basins — nodes and fragments were dragged starting from a circle layout.  
 Below: the 11 prototype basins — starting from a square layout.

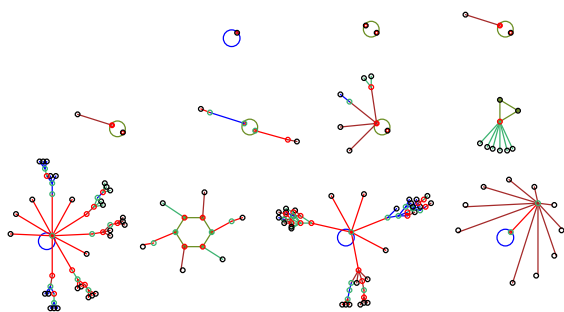


Figure 20.11: Inserting basins in the jump-graph for a 1d CA  $v2k3$ ,  $n=10$ , rule (dec)133. For a 1d or 2d CA, if edges were omitted in the jump-graph (section 20.3.1), the basin of attraction field will be compressed by default (section 26.2) to include just the prototype (non-equivalent) basins. Otherwise all the basins in state-space are shown.

Enter **i** in section 20.3.2 to redraw the basins centered at the jump-graph nodes positions, but without the jump-graph itself — suppressing 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.

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.

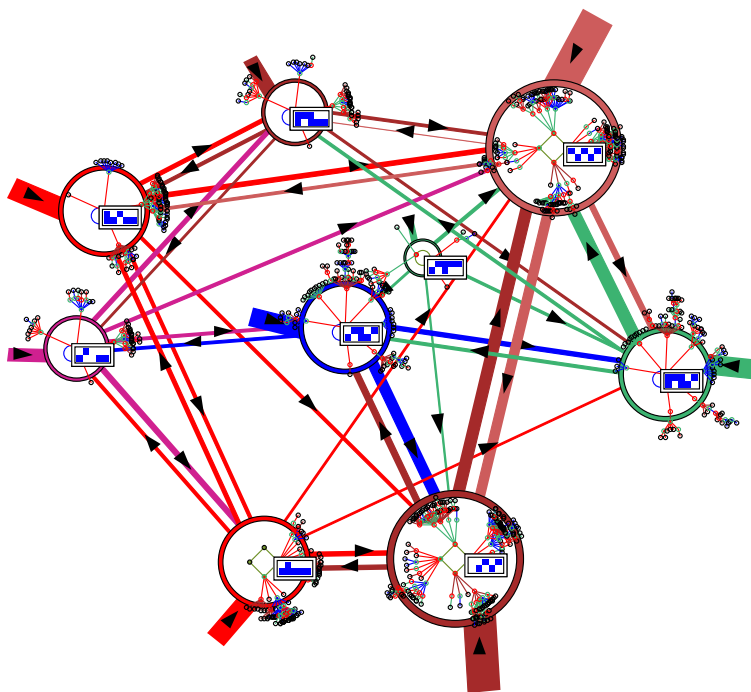


Figure 20.12: Basins of attraction redrawn at the jump-graph nodes, retaining the jump-graph itself. In this example one state in each attractor is also displayed. To create this figure as a vector PostScript file, the separate jump-graph and basin PostScript files were combined. This example is for the RBN  $v2k3$ ,  $n=10$ , defined in table 20.1.

### 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 “basin 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 basin 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 postScript (now p): greyscale-P color-p cancel-0: (for example)**

*if saving to PostScript is currently inactive*  
**save basin to postScript (now OFF): greyscale-P color-p: (for example)**

Enter **0** to deactivate, **P** or **p** to activate — a filename prompt will be presented (section 35.3). 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<sup>4</sup> 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 separate 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, 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 35.3). The default filename is `my_ngPS.ps` for a network-graph and `my_jgPS.ps` for a jump-graph. Once saving is complete, 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, 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 35.3). 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)**

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,

<sup>4</sup>For example, rename in a terminal, look at the PostScript file in “GhostView”.

**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 settings.

## 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 unreachable 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,

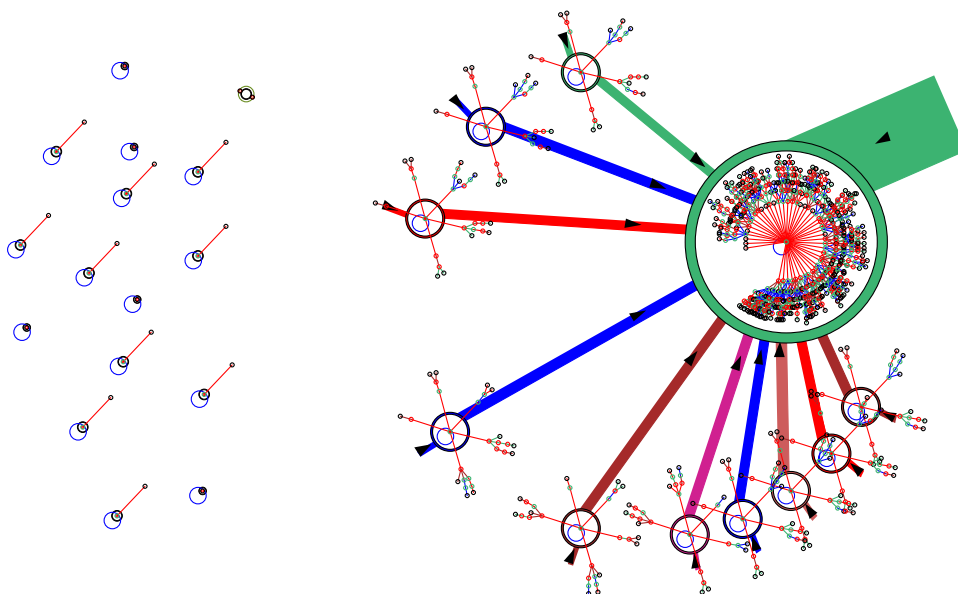


Figure 20.13: Disconnecting and isolating unreachable nodes. The isolated nodes are repositioned randomly on the far left side of the graph window. In this case with the default unreachable threshold of zero. This example is for a jump-graph of a 1d CA,  $v2k3$  rcode 104,  $n=10$ . The basins of attraction were redrawn at the nodes (section 20.7).

**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 labelled 1 to  $N$ , were  $N$  is the number of basins in the basin of attraction field. For each basin, the columns, also labelled 1 to  $N$ , show the number (or fraction of total) jumps to each basin. The meaning of the four further columns labelled **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 display the table in the terminal.

**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.

---

## 20.13 Space-time patterns within the network-graph

Space-time patterns can be shown running within the network-graph layout simultaneously to the normal 2d presentation described in chapter 32. Enter **g** at the space-time pattern interrupt/pause prompt (section 32.16) to set up the network-graph while space-time patterns are running — section 32.19 gives further details.

This allows enormous flexibility, as the network-graph has various default layouts: circle, spiral, 1d, 2d (square or hex), 3d, as well as allowing dragging nodes and components, and many other options for rearranging the graph (figures 32.37 — 32.38).

Once the network-graph space-time patterns are running, and the normal space-time presentation is in 2d, diagonal scrolling can be set with on-the-fly key `#`. To produce a scrolling image as in figure 20.14 or 4.9, suppress normal space-time patterns with on-the-fly key `J`.

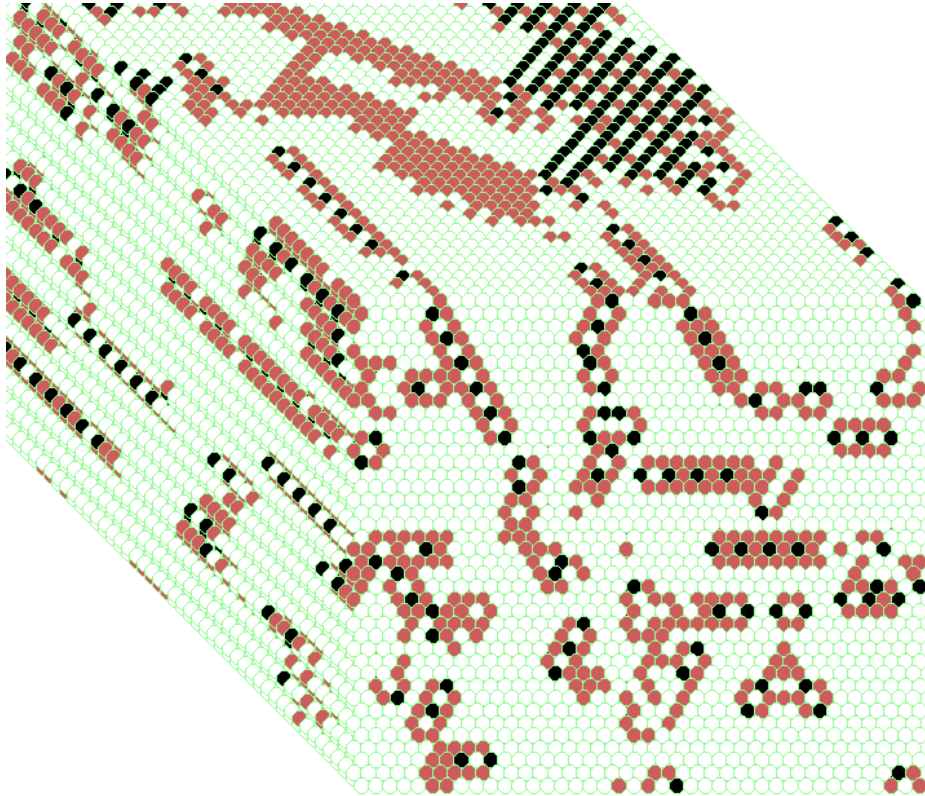


Figure 20.14: A 2d space-time pattern shown running within the network-graph layout, which has also been set to scroll diagonally. The present moment is the 2d pattern at the bottom-right.  
 2d CA  $40 \times 40$ , *v3k6*, *kcode(hex)* 0a502804958100.

---

# Chapter 21

## The Seed or initial state

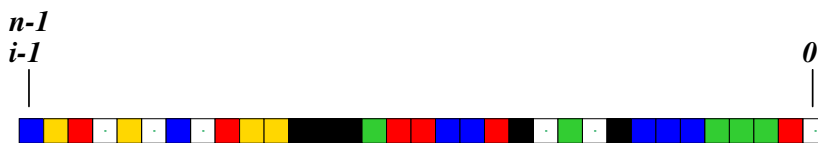


Figure 21.1: A 1d seed (initial state) is indexed as in section 10.2.1. This example is for a random seed,  $n=33$ ,  $v=6$ .

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<sup>1</sup>. This chapter describes how the seed is specified and amended<sup>2</sup>, including random “blocks”, drawing with the mouse, moving, manipulating and editing the presentation, and saving and loading the seed, or the image in vector PostScript. For 2d these methods can be applied to just a predefined “patch” – useful for designing and editing particle collisions for universal computation. Saving/loading can also be done in the “Golly” file format.

Section 10.2 described how the network is indexed and arranged in a regular 1d, 2d or 3d lattice (figures 10.5 – 10.7). 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 section 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).

---

### 21.1 The seed prompt

To set the seed a lower left window is displayed in the main prompt sequence. When resetting a seed while generating space-time patterns or basins, the window appears in the lower center-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** — for a random seed or central block with a given density (section 21.3). If a seed was previously set, the default is **bits-b** — for amending, or drawing the seed (section 21.4), manipulating a 2d patch, or saving/loading seeds or sub-seeds (sections 21.7 and 21.8) and vector PostScript images.

---

<sup>1</sup>Note that a seed is not required to generate a basin of attraction field.

<sup>2</sup>The methods are similar to “setting a single rule” in chapter 16

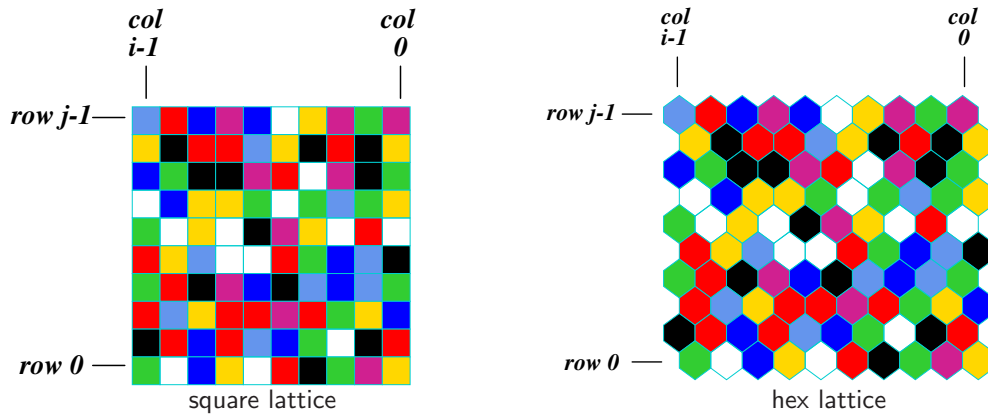


Figure 21.2: A 2d seed (initial state) is indexed as in section 10.2.2, showing both square and hex layouts. Divisions lines between cells shown here are light blue — the alternative is black, white, or none — where cells touch (figure 21.7). This example is for a random seed  $[i, j]=[10, 10]$ ,  $v=8$ .

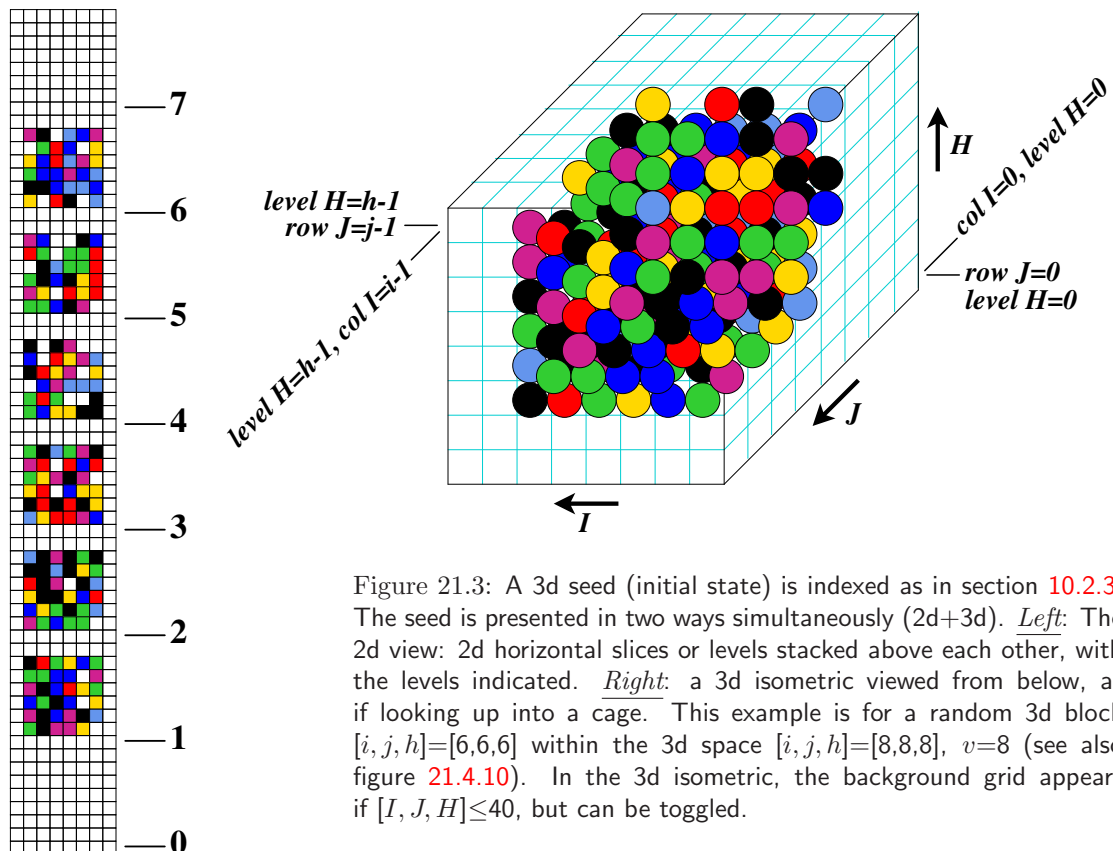


Figure 21.3: A 3d seed (initial state) is indexed as in section 10.2.3. The seed is presented in two ways simultaneously (2d+3d). *Left:* The 2d view: 2d horizontal slices or levels stacked above each other, with the levels indicated. *Right:* a 3d isometric viewed from below, as if looking up into a cage. This example is for a random 3d block  $[i, j, h]=[6, 6, 6]$  within the 3d space  $[i, j, h]=[8, 8, 8]$ ,  $v=8$  (see also figure 21.4.10). In the 3d isometric, the background grid appears if  $[I, J, H] \leq 40$ , but can be toggled.

*example for a 1d network*

Select SEED (v6 1d n=33), win-w empty-e fill-f prtx-x ascii-v  
rnd-r bits1d-b bits2d-B hex-h dec-d repeat-p load-l (def-r):

*example for a 2d network. Golly-G for v=2 only*

Select SEED (v2 2d ij=10,10), win-w empty-e fill-f prtx-x ascii-v Golly-G  
rnd-r bits2d-b hex-h dec-d 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 ascii-v  
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<sup>3</sup> in section 21.1 are summarized below. Further details are provided 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=5*)

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. The values are shown in the same order and layout as in figures 21.1, 21.2 and 21.3<sub>Left</sub>, so for 2d and 3d according to the dimensions as in table 21.1.

```
05d0697c89d20d8234169391d65f707cd2e441
5640645
7442351
0154043
2026447
1072627
6701746
4562101
```

```
0451eeb4821c69fb2d6bca2d6c0a7de1f570609465025e
21217
35322
02070
64773
```

```
13265
71213
26601
23736
```

Table 21.1: Examples of displaying  
2d and 3d seeds in the terminal.

*Left:* 2d 7×7, v=8.

*Right:* 3d 5×4×3, v=8.

```
07653
40602
24312
01136
```

<sup>3</sup>These methods are similar to setting a rule in section 16.2.



- ascii-v** ... to save or load the seed as an ASCII value string, a `*.see` file, described in section 21.10. This is useful for interchanging the seed between DDLab and alternative software.
- Golly-G**... (*for 2d binary networks*) to save or load the seed as an ASCII string in the Golly file format, a `*.g1y` file, described in section 21.11. This allows interchanging the seed between DDLab and Golly software ([hppt://www.conwaylife.com](http://www.conwaylife.com)).
- 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 continues.
- rnd-r** ... to set the seed at random or a as a central block, for 1d, 2d or 3d. There are various further options described in section 21.3, including setting the density, or the value bias — the proportions of different values.
- bits-b** ... to set the seed as bits or values (section 21.4) by “drawing” using the mouse or keyboard, and many other options depending on network dimensions as follows,  
**bits1d**: on a 1d graphic array for 1d networks. Subsequent prompts allows just a segment to be set (section 21.4.6).  
**bits2d**: on a 2d graphic array for 2d networks. This includes options for defining a patch which can be independently moved and manipulated (section 21.4.1)  
**bits2d+3d**: on a 2d view 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 (section 21.4.7), where the  $i, j$  aspect ratio can be changed (figure 21.8).
- 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** ... load a seed 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 or accepted in section 21.1, a seed, or just a central block of any size or shape, can be set at random, or with some bias. When resetting a seed or block on-the-fly in section 32.8.1, the central block size and the density-bias (and the value-bias) set here are respected. Firstly, a subsequent prompt is first presented to revise the default block size or to set an entire seed,

*for a 1d network,  $n=150$*

**central 1d block (def diam 30) all-a (max 150):**

*for a 2d network,  $88 \times 44$*

**central 2d block (def diam 17) rectangle-s all-a (max 88):**

*for a 3d network,  $60 \times 25 \times 40$*

**central 3d block (def diam 12) cuboid-s all-a (max 60):**

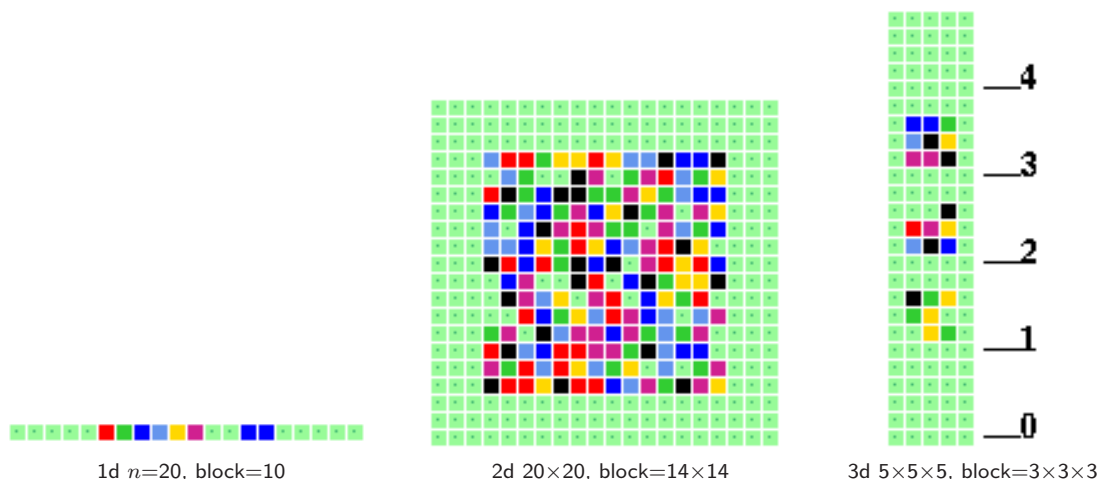


Figure 21.4: Examples of the graphic output when setting a seed at random (section 21.3), in this case a central random block, for 1d, 2d, and 3d — which is a 2d view of a 3d isometric similar to figure 21.3Left. Note that the “random” presentation is less versatile than “bits or values” (section 21.4).

The default central block size depends on the size and dimensions of the network and is a square or cube for a square or cubic lattice. Otherwise, for any network axis  $a$ , the default block axis  $b$  is set thus:  $a \leq 5$   $b=a$ ,  $a \leq 20$   $b=a/2$ ,  $a > 20$   $b=a/5$ . Enter **return** to accept the default. Enter a different size for the random block which applies all axes for a square or cube, but otherwise to the largest axis or axes only. Enter **a** for a random seed filling the whole network. For 2d or 3d, enter **s** to specify an exact size of a rectangle or cuboid – subsequent prompts are presented as follows,

*for a 2d network, 88x44*  
**rectangle: x(max 88, def 17):      y(max 44, def 8):**

*for a 3d network, 60x25x40*  
**cuboid: x(max 60, def 12):      y(max 25, def 5):      z(max 40, def 8)**

### 21.3.1 Seed at random — further options

The pattern is created at random and displayed in the seed window as a bit/value pattern<sup>4</sup> in 1d, 2d or 3d as in figure 21.4 according to the current density-bias of non-zero values which appears in a top-right window (section 21.3.2).

At the same time various changes and transformations<sup>5</sup> can be made (repeatedly) until **return** is entered to accept the seed. A subsequent reminder is shown in a small font,

*example for 2d*      (*for 1d rotate-l/r, for 3d rotate-l/r/f/b/u/d*)  
 tog-gaps-g exp/contr-e/c rotate-l/r/u/d another-n density-s/v comp-m back-q accept-ret

<sup>4</sup>The bits/values options in section 21.4 provide many extra presentation possibilities.

<sup>5</sup>These option are similar to setting a rule at random (section 16.3).

These options are summarized below — with more detail in the sections indicated.

- options ... what they mean*
- tog-gaps-g** ... to toggle gaps between successive blocks of 8 bits/values.
  - exp/contr-e/c** ... enter **e** to expand or **c** to contract the current scale of the bit/value graphic.
  - rotate-l/r/f/b/u/d** ... for 1d enter **l** or **r** to rotate the seed left or right by one cell. For 2d **l/r/u/d** rotates about the  $i, j$  axes, and for 3d **l/r/f/b/u/d** rotates about the  $i, j, h$  axes. Rotation is subject to periodic boundaries, a ring of cells in 1d, a torus in 2d, and a 3-torus in 3d.
  - another-n** ... for another random seed or block with the same density or value bias.
  - density-s/v** ... enter **s** to change the density-bias — initially values have equal probability (section 21.3.2). Enter **v** to set the value-bias (section 21.3.3) the proportions of different values in the seed.
  - comp-m** ... to toggle/transform the seed into its complement. For binary this changes 1s to 0s and vice versa. For  $v > 2$  each value  $a$  is changed to its complement  $a_c = (v - 1) - a$  (as in section 16.20 for rules).
  - 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 and/or block (expressed as a percentage) is displayed in a top-right density window. The default is an equal probability of each value, so 50% 1s for binary. The following density window example relates to figure 21.5(a),

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, or **S** for a 2d network in section 21.4.2, to change the density-bias<sup>6</sup> 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(b).

---

<sup>6</sup>The methods for seed density-bias are similar to rule density-bias (section 16.3.1).

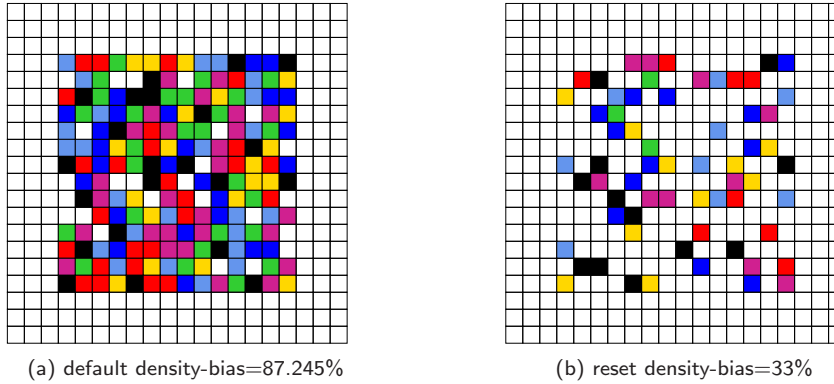


Figure 21.5: Setting the  $v=8$  density-bias for a central block  $14 \times 14$  within a 2d seed  $i, j=20 \times 20$ , showing the default and reset density. The graphic output would appear as in figure 21.4, but is presented here from “Setting the seed as bits or values” (section 21.4).

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, updating the density window, and when re-setting random seeds for space-time patterns on-the-fly (section 32.8.1).

### 21.3.3 Seed or block value-bias

2	35	65					
3	18	65	17				
4	12	12	65	11			
5	9	9	65	9	8		
6	7	7	7	65	7	7	
7	6	6	6	65	6	6	5
8	5	5	5	5	65	5	5
	0	1	2	3	4	5	6
% of each value							

Table 21.2: The default seed value-bias if not reset. The current seed value-bias can be randomly reset when running space-time patterns with on-the-fly keys `\` (backslash) for the whole seed or `/` (forward-slash) for a central block.

The frequency of values randomly assigned to a seed can be set for any distribution. If the frequency has not been set by the methods below, a default frequency will apply based on percentages of each value as in figure 21.2. The bias applies to the whole seed, but if a central block was defined in section 21.3, only that part of the biased seed is assigned.

Enter **v** in section section 21.3, or **V** for a 2d network in section 21.4.2, to set the proportions of different values in the seed or block, either as whole number percentages, or as actual numbers of each value if the seed size  $n \leq 255$ . The following series of prompts are presented in a top-right window<sup>7</sup>,

```

if S ≤ 255
bias random seed, %/value-p/v keep-k:
if S > 255
bias random seed, value-p keep-k:
if p was selected
0(100:45 1(55:4 2(51:4 3(47:4 4(43:9 5(34:9 6(25:18
if v was selected
0(120:56 1(64:7 2(57:7 3(50:7 4(43:12 5(31:12 6(19:15

```

Enter **k** to keep the last setting, otherwise enter **p** or **v** — then for each value, from 0 to  $v-2$ , enter the percentage or number — the value for  $v-1$  is set automatically. The availability for successive allocations of each value is shown with a bracket, for example at **3(50:** (for  $v=3$ ) enter a number  $\leq 50$ . If exactly 50 is entered the allocation will halt as complete. Entering a number  $> 50$  or **return** results in allocating 50 divided by the number of remaining values, so entering **return** repeatedly gives an even distribution.

A lower top-right window shows the resulting value-bias distribution for the example above, where “0–7” (for  $v=8$ ) indicates the order of values,

```

if p was selected
seed-bias: percent(100) 0-7: 7 18 9 9 4 4 4 45
if v was selected
seed-bias: numbers of values(120) 0-7: 56 7 7 7 12 12 15 4

```

If set, the value-bias will be maintained for further random seeds generated with *another-n* in section 21.3 updating the density window, and when re-setting random seeds for space-time patterns with on-the-fly keys \ (backslash) for the whole seed or / for a central block (section 32.8.1).

## 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, 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, and in a top-right inset window.

The colors correspond to the value colors (chapter 7) but 0s are initially colored light green. The seed can be reset to all-0s with **empty-e** or to any uniform value with **fill-f** in section 21.1 to provide a clean slate for setting new patterns. Alternatively, use this bit/value setting method for fine adjustments to a seed set by other methods.

<sup>7</sup>The methods for the rule value-bias in section 16.3.2 are similar.

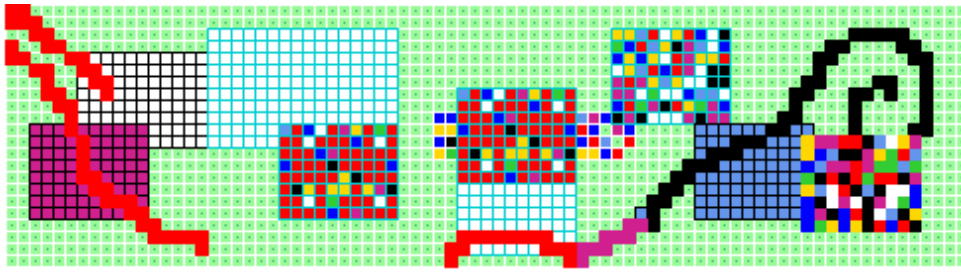


Figure 21.6: Playing with a 2d  $10 \times 8$  active patch in a 2d seed  $[i, j]=[88, 22]$ . The patch was set by the last two mouse clicks when inactive, and activated with `patch-p`. Many options (colored red in section 21.4.2) apply only to the patch, such as the presentation including the grid line color. Moving the patch with `l/r/u/d` leaves a blank trail with the current grid line setting/color, and stops at an edge. The patch has been jumped/copied with `j` to new pointer positions, and filled with colors and random patterns — if the jump would cross an edge it is truncated, and the smaller size is retained. Drawing with the mouse pointer or keyboard applies to the whole seed but can cross a patch. This example is a bitmap image to capture the grid line colors and presentations, which remain distinct while a patch is active, but become unified according to the last settings once the patch is deactivated with `stop-patch-p`.

### 21.4.1 2d patch options

New options have been added for 2d networks where a “patch” of any size can be defined and activated (enter `patch-p` in section 21.4.2) — thereafter many options (colored red in the prompt in section 21.4.2) apply only to the patch rather than the whole seed, for example the patch can independently be moved, edited, manipulated, filled with colors or random patterns, jumped/copied to any pointer position, saved/loaded as a `*.eed` file, or saved as a `*.ps` PostScript image.

These features have been added in order to make it easier to design, edit and test spacial patterns for rules such as the game-of-Life [10], the spiral-rule [45], and the X-rule [13], where particle collisions, gliders and static structures can combine to create glider-guns and emergent configurations of greater complexity to implement logical gates for universal computation.

### 21.4.2 Seed: bits/values reminder window

Once the seed is displayed as bits or values, many possibilities become available for amending the seed, including drawing with the mouse and keyboard<sup>8</sup>. A top-right reminder window, and an inset, show reminders of the various options and current settings (summarized 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: set 1s width=1`

*the inset with current settings* ↘

**keys:** `val/draw-(1-0) vert-v move-arrows` `1d I=149, v=2 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 axis-[/]`

**rotate-l/r/+/- flip-X comp-m back/cont-q/ret**

<sup>8</sup>Similar methods apply for the rule in section 16.4

2d seed,  $v=8$ , 2d seed  $i, j=44 \times 44$

togpatch inactive — define a patch here with the last 2 mouse clicks

**keys:** val/draw-(7-0) vert-v move-arrows 2d IJ=44x44 val=7 scale=5 rot=1  
**mouse:** move-click draw-drag width-w pan-U/D PScript-P file-F  
**tog:** gaps/0color/dots/divs/divcolor-g/^\./i/! redef-S/V exp/contr-e/c  
 patch-p rot-l/r/u/d/+/- spin-s flip-X/Y comp-m toghex-x back/cont-q/ret

2d seed with patch activated,  $v=8$ , 2d seed  $i, j=44 \times 44$

options in red apply to the patch only — deactivate with **stop-patch-q** to redefine the patch

**keys:** val/draw-(7-0) vert-v move-arrows 2d IJ=44x44 val=7 scale=5 rot=1  
**mouse:** move-click draw-drag width-w patch active: 22,22-29,26  
**tog:** 0color/dots/divs/divcolor-^\./i/! fill-f rnd-S/V PScript-P file-F exp/contr-e/c  
 stop-patch-q move-j/l/r/u/d/+/- spin-s flip-X/Y comp-m toghex-x cont-ret

3d seed,  $v=8$ ,  $i, j, h=15 \times 15 \times 15$

**keys:** val/draw value 7-0, vert-v 3d IJH=14x14x14, scale=5 rot=1  
**mouse:** move-click draw-drag width-w move-arrows pan-U/D PScript-P file-F  
**tog:** gaps/0color/dots/divs/divcolor-g/^\./i/! exp/contr-e/c/E/C  
 rotate-l/r/b/f/u/d/+/- spin-s flip-X/Y comp-m grid3d-t back/cont-q/ret

### 21.4.3 Seed: bits/values options summary

The bit/value options are summarised below, and some are described in detail in later sections. If a patch is active in a 2d, options colored red apply just to the patch, otherwise to the whole seed. To redefine a 2d patch the patch must be inactive.

options ... what they mean

**keys: val/draw 7-0** ... enter a valid number to select the value/color between 0 and the  $v-1$ . Keep the key pressed to draw a horizontal line.

**vert-v**... keep **v** pressed to draw a vertical line downwards in the selected value/color.

**move-arrows** ... all four arrow keys move the cursor around the seed (up/down arrows for multiple rows).

**mouse: move-click** ... left click on the bits/values 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 pointer with the left mouse button pressed — release the button to stop. The right mouse button draws the complementary value/color (section 16.20) in the same way.

**width-w** ... enter **w** to change the width (initially 1 cell) 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 view 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 16.4, for rule patterns, gives examples).

**U/D** ... (for 2d and 3d only) to pan a 2d seed graphic up and down. 2d will pan by 1/10th of the total height  $j$ . The 2d view of 3d (figure 21.3) will pan by 1 level. This allows different parts of large seeds to be brought into focus for setting bits.

**PScript-P** ... to save the seed or 2d patch as a vector PostScript image (sections 21.4.8, 21.4.11).

**file-F** ... to load/save the seed or 2d patch (section 21.4.8).

**tog: gaps-g** ... (not if a 2d patch is active) enter **g** to toggle gaps between successive blocks of 8 bits/values (as in figure 16.3 for rules). Active gaps prevent a 2d patch from activating.

**tog: Ocolr-^** ... to toggle the zero value color between light green and white for the seed or 2d patch (figure 21.9).

**tog: dot-** ... enter a dot to toggle a dot at the center of each zero cell for the seed or 2d patch (figure 21.11).

**tog: divs-i/!** ... enter **i** to toggle division lines on/off between cells for the seed or 2d patch. Enter **!** (exclamation mark) for a 3-way toggle of the division line color — white, black, and light blue (figure 21.7).

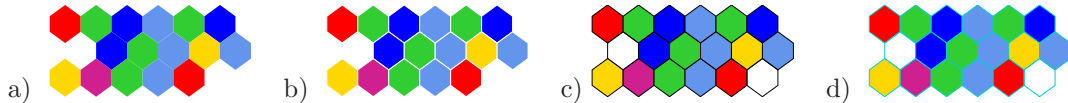


Figure 21.7: Changing the division line presentation/color between cells: a) no lines — cells touch. b) white. c) black. d) light blue. 2d hex layout  $6 \times 3$  (for a square layout see figure 32.14).

**fill-f** ... (for a 2d active patch only) enter **f** to fill the patch with the selected color, which can be reselected first if necessary by first entering a valid number between 0 and the  $v-1$ .

**rnd-S/V** ... (for 2d only) If a patch is active, to fill the patch with a random pattern by two methods: **S** by the current density-bias (section 21.3.2), or **V** by the current value-bias (section 21.3.3). If the 2d patch is inactive, **S** and **V** redefine the density-bias or value-bias with prompts as in sections 21.3.2 and 21.3.3.

**exp/contr-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 view of 3d, which are treated separately.

**axis-[/]** ... (for 1d only) to change the aspect ratio presentation, enter **[** to halve the x-axis, **]** to double the x-axis, the y-axis will change accordingly.

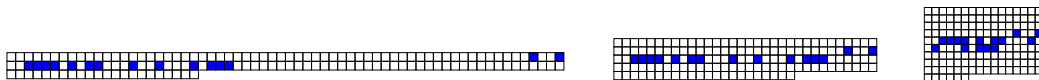


Figure 21.8: Changing the 1d seed aspect ratio presentation with key-hits **axis-[/]**,  $n=150$ .



- patch-p** ... (for 2d only) enter **p** to activate the patch — the last two mouse clicks (in any order) will define the top-left and bottom-right corners of a new patch. The seed reminder window (21.4.2) will change accordingly.
- stop-patch-q** ... (for a 2d active patch only) enter **q** to deactivate the patch. The seed reminder window (21.4.2) will change accordingly.
- rot-l/r/b/f/u/d/+/-** ... The seed can be “rotated” about the periodic axes in 1d, 2d and 3d by the current rotation interval — default 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 increase or decrease the rotation interval — shown in the current settings inset.
- move-j/l/r/u/d/+/-** ... (for a 2d active patch only) **move-j** jumps/copies the patch, see below. **l/r/u/d** moves the patch left/right/up/down by the current move interval — default one cell. The patch will leave a trail of empty (zero) cells with the current cell presentation, and will stop at an edge. The current active patch position, top left and bottom right coordinates, are shown in a top-right inset. Enter **+** or **-** to increase or decrease the move interval — shown in the current settings inset.
- move-j** ... (for a 2d active patch only) Enter **j** to jumps/copy/activate the patch to the current cursor position — which will be the bottom-right corner. The pre-active patch remains in place. If the jump would cross an edge it is truncated, and the smaller size is retained for future jumps or moves.
- spin-s** ... (for 2d and 3d only) For 2d where  $i = j$  the seed or patch is spun clockwise by  $90^\circ$  so 4 spins returns to the start. Similarly for 3d where  $i = j$ , each level  $h$  is spun clockwise by  $90^\circ$ . If  $i \neq j$  a square part including  $[I, J]=[0,0]$  will be spun.
- flip-X/Y** ... enter **X** for a horizontal flip or reflection of the seed or 2d patch. For 1d this reflects the  $i$  axis (figure 18.2), 2d and 3d each  $i$  axis is reflected. For 2d and 3d enter **V** for a vertical flip of each  $j$  axis of the seed or 2d patch.
- comp-m** ... to toggle the seed or 2d patch into its complement. 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).
- toghex-x** ... (for 2d only) to toggle between a square and hexagonal grid.
- grid3d-t** ... (for 3d only) to toggle grid lines, and update the 3d isometric — after drawing bits/values on the 2d view.
- back/cont-q/ret** ... enter **return** to accept the seed and continue, **q** to backtrack. If a 2d patch is active, the prompt is just **cont-ret** — **stop-patch-q** will deactivate the patch as noted above.

### 21.4.4 Seed: bits/values current settings inset

`3d IJH=14x14x14 val=4 scale=5 rot=1` ← the inset with current settings, for 3d

*current settings ... what they mean*

IJH= ... the current cursor 3d coordinates, or IJ= for 2d, I= for 1d.

val= ... the current drawing value/color.

scale= ... the current scale of cells in pixels, which can be expanded and contracted.

rot= ... the current interval by which the seed can be rotated, or a 2d patch moved.

### 21.4.5 Seed: setting bits/values with the keyboard and mouse

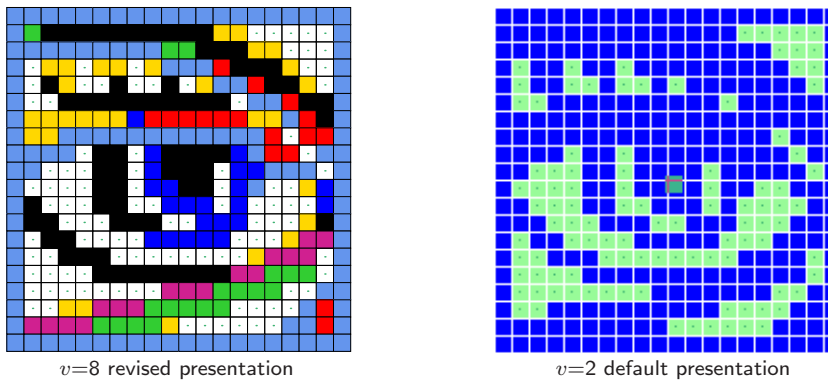


Figure 21.9: Drawing bits or values on a 2d seed  $[i, j]=[20, 20]$ . *Left*: The  $v=8$  “eye” was drawn with the mouse and keyboard. *Right*: the default presentation of the eye for  $v=2$ ; light green 0s and white divisions; the flashing cursor was moved to the center. The  $v=8$  eye was loaded into a  $v=2$  base seed.

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 cursor appears only on the 2d view of 3d. Its position is displayed in the top-right inset window (section 21.4.4). The cursor is repositioned with the mouse by clicking either the left or right button on the new position<sup>9</sup>, 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 view of 3d — the 3d isometric 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 cell. 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 pointer anywhere over the seed with the left button pressed — release to stop. To draw the complement of the active value (section 16.20)

<sup>9</sup>Mouse behavior differs slightly between Linux-like systems and DOS. In Linux the mouse pointer changes direction within the bit/value 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 pointer is confined within the pattern and clicking the left or right buttons sets values as well as repositioning the flashing cursor.

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. A button press outside the grid area changes the reminder window to `outside grid`.

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 seed is also displayed in decimal (if applicable — section 21.6).

### 21.4.6 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 allow 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 centered:** (*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.

### 21.4.7 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 21.4.8).

### 21.4.8 Setting bits/values: filing and PostScript

The seed (or a 2d patch) 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 21.4.2 — both options follow similar methods, and are similar to the rule options in section 16.4.4. File extensions and default filenames (chapter 35) are as follows,

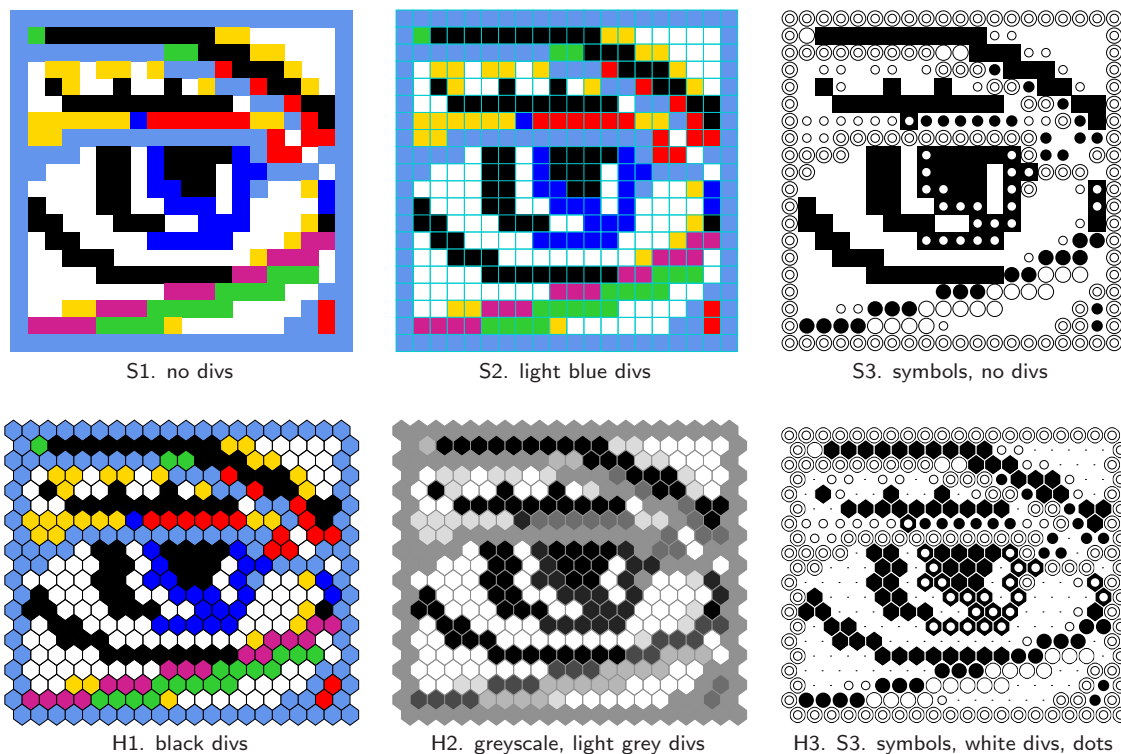


Figure 21.10: Examples of PostScript 2d seed output, the “eye” as in figure 21.9,  $[i, j]=[20, 20]$ ,  $v=8$ . A variety of options are available: color, greyscale, symbols, hex, square, division color, and zero dots. Top Row: square layout. Bottom Row: hex layout.

PostScript files ... \*.ps files: for 1d, 2d, and the 2d view of 3d, (as in figures 21.1), 21.2 and 21.3 Left) — default filename `my_sps.ps`.  
 for the 3d image only (as in figure 21.3 Right) — default filename `my_3dps.ps`.

seed files ... \*.eed — default filename `mysee_vv.eed` where  $v$  = value-range.

One of the following prompts appears in a top-right window <sup>10</sup>,

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:

<sup>10</sup>If a 2d patch is active these prompts are dealt with directly — for PostScript the next prompt is in section 21.4.10 — then the relevant filing prompt appears as in section 35.3

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 summarized below,

- load-l** ... load a seed file (section 21.7). As many seed files as required can be loaded into the current seed. Differing value-ranges, dimensions and sizes are possible, subject to the constraints listed in sections 21.7.1 — 21.7.5.
- save-a** ... save the seed, as a seed file or PostScript. For PostScript and 3d, **save-a** applies to the 2d view of the 3d image, as in figure 21.3.
- save patch-p** ... 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 21.4.9.
- only 3d-3** ... (for PostScript of 3d only) save a PostScript file of the seed 3d isometric (section 21.4.11).
- 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 21.4.10). Files are loaded and saved from the filing prompt (section 35.3).

### 21.4.9 Setting bits/values: saving a patch

Part only<sup>11</sup> of the seed (in 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 21.4.8, one of the following set 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 at any position within the same or another seed (section 21.7). Note that for 2d networks the patch options in section 21.4.1 provide alternative methods for saving a patch. The facility to save and load a patch, combined with the patch options for 2d in section 21.4.1 provide very flexible methods for editing the seed.

---

<sup>11</sup>If a 2d patch is active it can be save directly – enter **P** or **F** in section 21.4.2.




















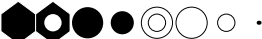

$v$	values	color hex	symbols hex	symbols square
2	10			
3	210			
4	3210			
5	43210			
6	543210			
7	6543210			
8	76543210			

Figure 21.11: Symbols for PostScript seed output for  $v = 2$  to 8. Enter **s** in the PostScript prompt (section 21.4.10) to show values as symbols replacing colors. Symbols come in two versions, hex and square, depending on the bits/values display in section 21.4.3 which is toggled with **x**. The value zero is shown here as a dot, but this can be blank or the dot size changed.

### 21.4.10 Setting bits/values: PostScript prompt

for 1d, 2d, and the 2d view of 3d as in figure 21.3 Right

When creating a PostScript image of a seed there are a variety of presentation possibilities as illustrated in the figures in this chapter (21.1 — 21.13) and other figures in the manual. Note that the following options do not apply to a PostScript 3d image as in figure 21.3 Right, which follows its own methods (section 21.4.11).

Enter **a** (or **p** for part of the seed) at the prompt in section 21.4.8 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<sup>12</sup>, for example,

```
create PostScript image for 2d, 20x20 (or 1d, 3d)
symbols-s greyscale-g color-c direct-d exit-q (def-c): color-c/C for v=2)
```

These options are summarized below (then subsequent options continue),

*options ... what they mean*

**symbols-s** ... to show values as symbols (figures 21.10, 21.11).

**greyscale-g** ... to show values in greyscale (figures 21.10).

**color-c** ... (the default) to show values in color.

**color-C** ... (for binary  $v=2$  networks) by default 1s are colored — blue on a white background and yellow on black background. Enter **C** to instead save 1s in black or white respectively.

<sup>12</sup>From the space-time pattern pause/interrupt prompt (section 32.16) 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 21.4.10. For 1d space-time patterns, **P** gives the direct option only.

**direct-d** ... (only when interrupting space-time patterns, section 32.16) to scan cell colors directly from a 1d space-time pattern, or a snapshot of 2d, or the 2d view of 3d. If **direct-d** is selected the second line is replaced with,

**direct: greyscale-g color-c (def-c): (symbols-s does not apply)**

For interrupted 1d space-time patterns only, if **direct-d** was selected above, the next two prompts limit the part of the image to be saved (section 32.18)

**1d size (max/def=150): time-steps (max/def=669):**

Once these options are complete, the prompt to amend other settings is presented, for example,

**cellscale=5.00 dots(on)=0.70 divs(g), 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,w,b,g):**

Enter changes required or **return** to accept defaults, which follow the current bits/values presentation. The options are summarized below,

options ... what they mean

**cellscale** ... enter a new cell width in pixels.

**togdots-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,w,b,g)** ... the initial default division (color) depends on the bits/values presentation, and is shown in the prompt, for example **divs(off)** means that there are no divisions and adjoining cells touch. To change, enter **0** for off, **b** for black, **w** for white, and **g** for light blue/grey. The new choice becomes the default.

Cells with value zero (0color) are initially colored light green in the bits/values presentation (section 21.4), which can be toggled to white with *0color-^* in section 21.4.2 – the PostScript file will follow the current 0color.

Once these choices are complete, the \*.ps file is saved from the filing prompt (section 35.3). Section 36.1 explains how to view, edit, and crop the PostScript image.

### 21.4.11 Setting bits/values: PostScript 3d image

*for the 3d image as in figure 21.3 Right*

For a 3d network (or space-time pattern) if *PScript-P* is selected in section 21.4.8 the following top-right prompt is presented

**PostScript 3d: save all 2d-a, save patch 2d-p, only 3d-3: (for 3d)**

To save all or part of the 2d view of the 3d seed as a vector PostScript file (default filename `my_sPS.ps`), enter **a** or **p** — further prompts are presented described in section 21.4.10.

Enter **3** to save the 3d image to a vector PostScript file at the current scale (default filename `my_3dP.ps`) — the following top-right prompt is presented,

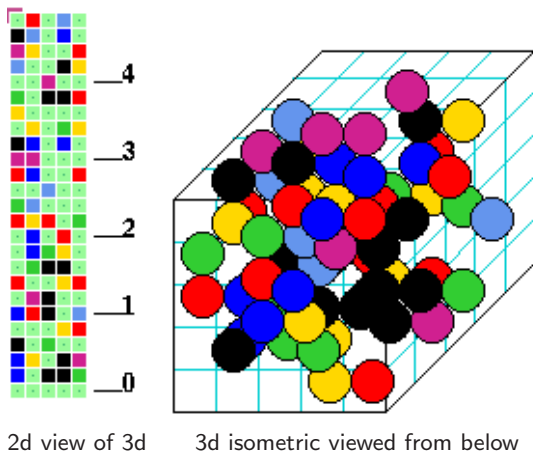


Figure 21.12: An example of a 3d bit/value seed,  $v=8$ ,  $n=5 \times 5 \times 5$ , as it appears with the prompt in section 21.5. The seed is shown as both a 2d view and a 3d isometric. Drawing bits/value applies only to the 2d view — but both 2d and 3d can be rescaled independently. This figure is bitmap graphics, not vector PostScript, but the 2d and 3d images can be captured separately in vector PostScript and combined as in figures 21.13, 21.3 and 32.18.

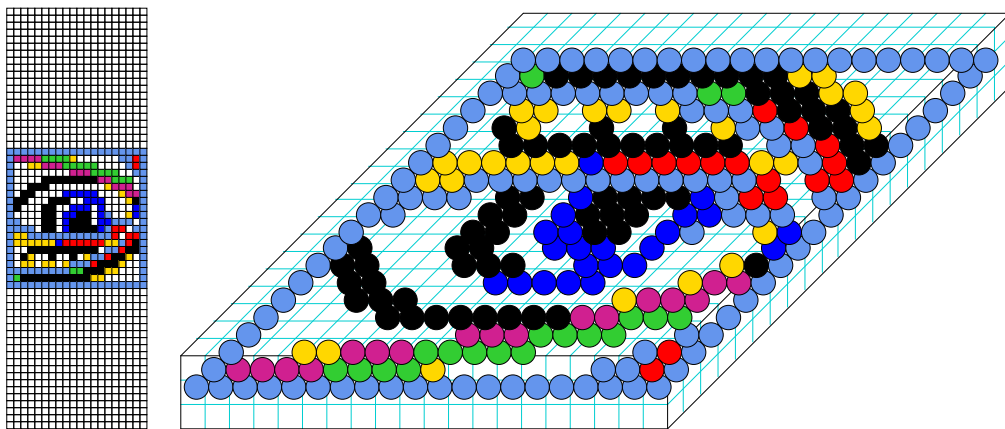


Figure 21.13: The  $v=8$  2d  $20 \times 20$  “eye” in figure 21.9 was loaded into a shallow 3d seed  $20 \times 20 \times 3$ , then flipped upside-down with **V**. The result is shown as it appears simultaneously in the 2d view and in the 3d isometric.

### 3d to PostScript 3d: greyscale-**P** color-def:

Enter **P** for greyscale, or **return** for the same colors as the 3d seed image.

Finally, a top-right filing prompt (section 35.3) is presented for the vector PostScript \*.ps filename. 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.





Figure 21.14: 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 presentation, allowing the seed to be reconfirmed or amended.

During the hex setting procedure, a top-right reminder window displays the following,

```
enter hex, arrows-move 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 the seed will be presented as bits/values displayed in 1d, 2d or 2d+3d (section 21.4), where it can be reconfirmed or amended.

## 21.6 Setting the seed in decimal

*applicable only for a limited range of  $v$  and system size  $n$*

$v$	max- $n$	max decimal
2	32	4294967295
3	20	3486784400
4	16	4294967295
5	13	1220703124
6	12	2176782335
7	11	1977326742
8	10	1073741823

Table 21.3: Seeds in decimal — network size limitations — the maximum size of a seed  $n$  for each value-range  $v$ , and the corresponding maximum decimal number.

The decimal option remains valid as long as the decimal equivalent of the bit/value string representing the seed is within the limits<sup>13</sup> in table 21.3.

If **d** is an available option and 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,

<sup>13</sup>The same limits apply when setting a rule in decimal (table 16.2 section 16.6).

**decimal seed, enter 0-4782968 (def-rnd):5555555 - too big! back-ret:**

Enter **return** to revert to the seed prompt. Once successfully selected, the decimal seed is presented again as bits/values (section 16.4), where it can be reconfirmed or amended.

## 21.7 Loading a seed, and loading constraints

Enter **I** in section 21.1, or **F** followed by **I** in section 21.4.8 to load a seed (.eed file)<sup>14</sup>.

When loading (from the top-right filing prompt, section 35.3) DDLab will know the *file* network parameters — ideally these would be the same as the pre-existing *base* network parameters — however, differing value-ranges, dimensions and sizes are permissible, subject to the constraints listed in sections 21.7.1 — 21.7.5. As many seed files as required can be loaded into the current base, and positioned anywhere if the base is bigger than the file.

### 21.7.1 loading a seed — all parameters equal

If all file-parameters, value-range  $v$ , dimension 1d, 2d or 3d, and size  $[i, j, h]$ , are the same as all base-parameters, the file is loaded without further ado.

### 21.7.2 loading a seed — unequal value-ranges

The value-range  $v$  can differ between file and base, but base- $v$  will remain as the current value-range. For file- $v <$  base- $v$ , loading is not a problem. For file- $v >$  base- $v$ , any file-value that exceeds maximum base- $v$  will be reduced to maximum base- $v$ . In either case a top-right warning similar to the following is presented, coming before any other prompt,

**file-v(2) != base-v(8), abort-q, load anyway-ret:** (*for example*)

### 21.7.3 loading a seed that fits within the base

If the seed-file (with file- $dim \leq$  seed- $dim$ ) fits within the base, 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<sup>15</sup>, for example,

*example for a 1d network*

**1d: i=100, file size=20 abort-a or enter start pos  
(def 40, max 80, rnd-r) i:**

*example for a 2d network*

**2d:i,j=100,100 file:i,j=6,6 abort-a or enter start coords  
(def 47,47, max 94,94, rnd-r) i: j:**

*example for a 3d network*

**3d:i,j,h=200,100,100 file:i,j,h=44,44,44 abort-a or enter start coords  
(def 78,28,28 max 156,56,560 rnd-r) i: j: h:**

<sup>14</sup>Saving a seed was described in sections 21.4.8 and 21.8 — saving just part of a seed, a patch, in section 21.4.9.

<sup>15</sup>This is similar to loading a sub-network in section 19.4.3.

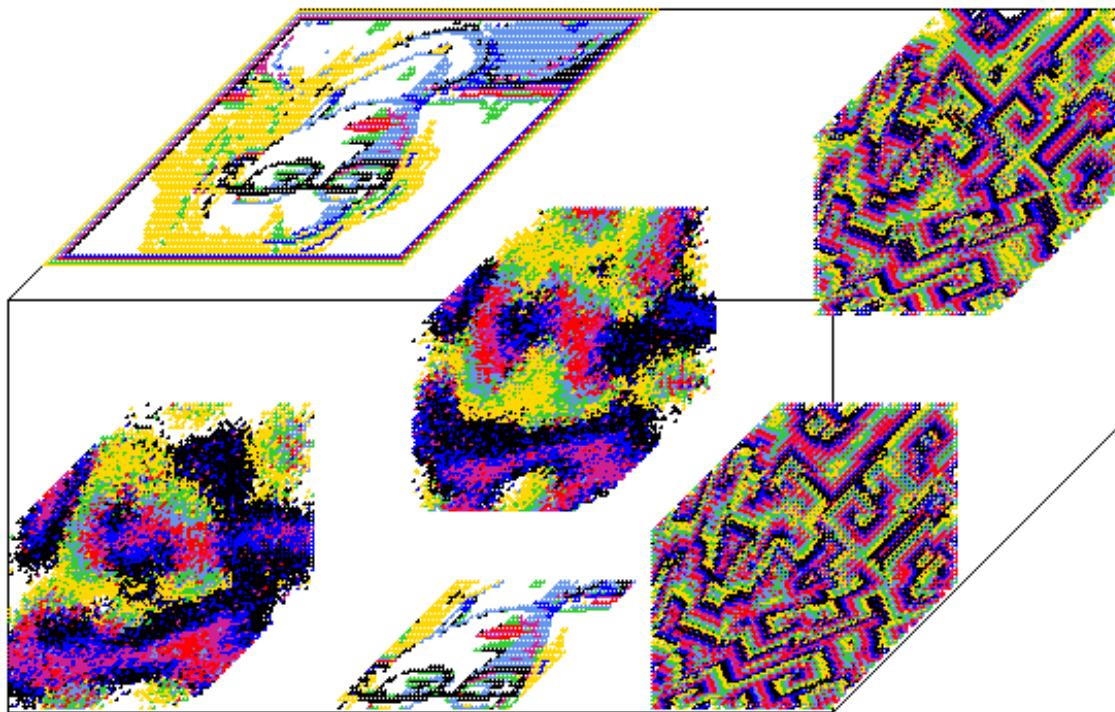


Figure 21.15: Loading seeds that fit within the base. Any number of seeds can be loaded positioned anywhere. This example includes 3d seeds from figure 13.4 (and variations), and 2d seeds from figures 2.3 and 21.16 loaded into a 3d  $200 \times 100 \times 100$  base.

Enter the coordinates in the base-seed to locate the file-seed's lower right cell, **return** for a default central position, or **r** for a random coordinate. Any number of file seeds can be loaded.

If  $\text{file-dim} < \text{seed-dim}$  (and the file-seed fits within the base), the smaller file dimension is increased to match the base by adding axes with coordinates of size 1. For file-2d, axis  $h=1$  is added to load into base-3d. For file-1d, axis  $j=1$  is added to load into base-2d, and axes  $[j, h]=[1, 1]$  to load into base-3d. File-axes of size 1 in the top-right prompt indicate that  $\text{file-dim} < \text{seed-dim}$  on these axes. For example, the prompt below indicates a 1d file-seed,

*example of 2d into 3d*

```
3d:i,j,h=66,66,66 file:i,j,h=11,11,1 abort-a or enter start coords
(def 27,27,32 max 55,55,64, rnd-r) i:    j:    h: (for example)
```

*example of 1d into 3d*

```
3d:i,j,h=33,11,22 file:i,j,h=14,1,1 abort-q or enter start coords
def 9,5,19 max 19,10,21: i:    j:    h: (for example)
```

#### 21.7.4 loading a seed that does not fit within the base, 2d and 3d

A seed-file that is bigger than the base on one or more axes, or a seed-file where  $\text{file-dim} > \text{seed-dim}$ , can still be partially loaded, but there is no option to locate the file.

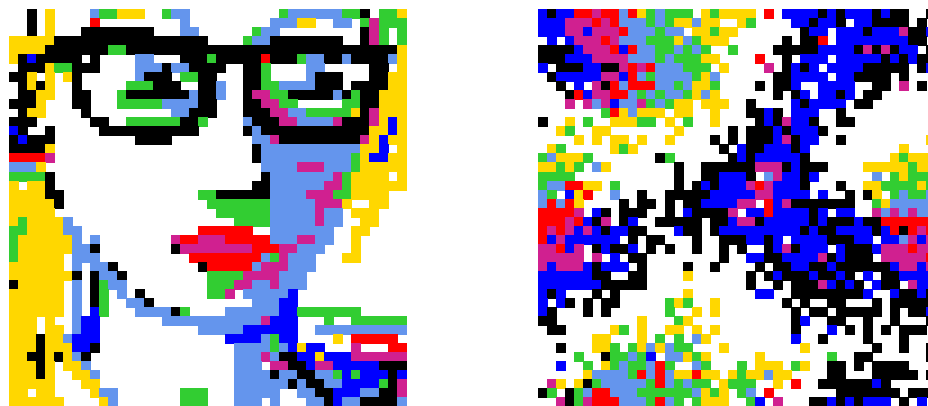


Figure 21.16: *Left*: Loading a 2d seed file,  $88 \times 88$ , the “portrait” from figure 2.3, into a 2d base network  $44 \times 44$ . *Right*: Loading a 3d seed,  $44 \times 44 \times 44$ , the snapshot,  $v=8$ , from figure 13.4 *Right* into a 2d base network,  $44 \times 44$  — just the central level is loaded.

For 2d and 3d seed-files and bases, a central zone of the file will be automatically selected and loaded into a central zone in the base. A top right prompt will indicate the dimensions (base-*dim* and file-*dim*) and if any file coordinates are bigger than the seed coordinates, for example,

```
basedim=filedim 2=2, coords bigger (for 2d)
2d:i,j=44,44 file:i,j=88,88 abort-q, load anyway-ret:(for example)
```

When loading a 3d file into a 2d base, just the central horizontal level will be loaded centrally, with the following top-right prompt,

```
basedim;filedim 2<3, filedim reduced (3d into 2d)
2d:i,j=44,44 file:i,j=44,44 abort-q, load anyway-ret:(for example)
```

When loading a 2d file into a 3d base, if the 2d file is bigger than the base on any axis, the central part of the 2d file will be placed at the central horizontal level of the 3d file, with the following prompt,

```
basedim;filedim 3<2, filedim reduced, coords bigger (2d into 3d)
2d:i,j=7,7,7 file:i,j=15,11,1 abort-q, load anyway-ret:(for example)
```

Note that the 2d file dimensions have been increased to 3d by adding a vertical axis of size 1, as in section 21.7.3.

### 21.7.5 loading a seed that does not fit within the base, 1d

In the case of a 2d or 3d seed file to be loaded into a 1d base, or a 1d seed file to be loaded into a 2d or 3d base where the  $i$  base coordinate is smaller than the  $i$  file coordinate, the entire string, as much as will fit, will be loaded according to the equivalent 1d indexing, from 0 to  $n-1$ . A top-right prompt is presented, for example,

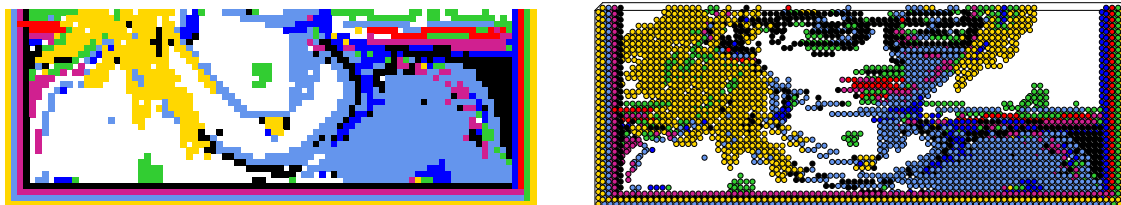


Figure 21.17: Loading a 1d seed file into a 2d and 3d base network,  $v=6$ . The entire string, as much as will fit, will be loaded according to the equivalent 1d indexing, from 0 to  $n-1$ . The 1d file-seed is the “portrait” from figure 16.15 where  $n=7744$  ( $88\times 88$  when presented in 2d). The 2d file was loaded *left*: into a 2d base  $88\times 33$ , and *right*: into a 3d base  $88\times 2\times 33$ .

```

basedim;filedim 1<3, filedim reduced (3d into 1d)
1d:i=45, file:729 abort-q, load anyway-ret: (for example)

basedim;filedim 2<1, filedim increased, coords bigger (1d into 2d)
2d:i,j=40,40 file:i,j=900,1, abort-q, load anyway-ret: (for example)

```

---

## 21.8 Saving a seed

Once the seed has been accepted, a top-right window shows the density of non-zero bits/values (section 21.3.2) and gives another chance to revise or save the seed,

```

density 3-1s=303/400=75.750%, bias=75.000% (example for v=4)
SEED: revise-q save-s cont-ret

```

Enter **q** to revise, or **s** to save the seed (as a `.eed` file<sup>16</sup>) from the top-right filing prompt (section 35.3). If just **return** is entered, the seed will be saved automatically with the default filename `mysee_vv.eed` where  $v$  is the value-range.

The seed, or part or a seed, can also be saved from the reminder window when setting bits/values. In section 21.4 enter **F** followed by **a** to save the whole seed, or by **p** to save just a patch as described in section 21.4.9

---

## 21.9 Seed file encoding

The `.eed` seed file is encoded<sup>17</sup> starting with either 5 or 9 leading bytes depending on the network size  $n$ , 5 lead bytes for “small”  $n\leq 65534$ , or 9 lead bytes for “big”  $n\geq 65535$ . The leading bytes define  $v$ ,  $n$ ,  $(i, j)$  the axes of a 2d network, and  $(i, j, h)$  the axes of a 3d network. For a “small” network,  $n$  requires 2 bytes, and  $(i, j)$  requires 1 bytes each. For a “big” network,  $n$  requires 4 bytes, and  $(i, j)$  requires 2 bytes each. for 3d  $h$  is implicit in  $n$  and  $(i, j)$ .

The encoding proceeds as follows,

---

<sup>16</sup>Seed files in multi-value versions of DDLab and the old binary versions prior to 2002 are incompatible.

<sup>17</sup>See also the file encoding for single rules (section 16.16) and the wiring/rulemix (section 19.3).

byte 0 ... value-range  $v$  ( $k$  is not relevant in a seed file).  
 byte 1,2 (big  $n$  byte 1,2,3,4) ... network size,  $n$ .  
 byte 3,4 (big  $n$  byte 5,6 and 7,8) ...  $[i, j]$  which gives the dimensions and axis sizes, If  $i = 0$  and  $j = 0$  the seed is 1d. If both  $i > 0$  and  $j > 0$  the seed can be either 2d or 3d.  $h = n/(i \times j)$ . If  $h > 1$  the seed is 3d, otherwise 2d.  
note axes limits ... for “small”  $n$  max axes ( $i, j, h$ ) must be  $\leq 255$ , for “big”  $n$  max axes ( $i, j, h$ ) must be  $\leq 65535$ . These limit are enforced when setting the axes (sections 11.6, 12.3.1).

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 or 9 leading bytes plus the bytes for the seed itself  $R$ , where  $R = \lceil \frac{n \times v_{bits}}{8} \rceil$  bytes, the upper absolute value, minimum 1 byte.

## 21.10 Saving/Loading a seed as an ASCII string

Enter *ascii-v* at the seed prompt in section 21.1 to save or load the seed as an ASCII value string, a *\*.see* file, (see Filing chapter 35). This is different from a binary seed file (section 21.9), but is useful for interchanging the seed between DDLab and alternative software.

The following top-right prompt is presented,

**ASCII seed (v=8 n=56) load/save-l/s:** (*for example*)

When loading, if *file-v* is not equal to *base-v*, the following top-right notice is displayed,

**file-v(8) != base-v(3), abort-q, load anyway-ret:** (*for example*)

and if *file-size* is not equal to *base-size*, the following top-right notice is displayed,

**file-size(44) != base-size(56), abort-q, load anyway-ret:** (*for example*)

In either case, enter **q** to abort loading, or **return** to continue loading. Although the loading prompt warns of a conflict between the file and base (value-range  $v$ , or size  $n$ ) with an option to abort, any ASCII file within DDLab’s naming constraints (section 35.1) with a *\*.see* extension can nevertheless be loaded — as much or as little as will fit, starting at rule-table index zero. Any *file-v* greater than *maxbase-v* will be reduced to *maxbase-v*.

### 21.10.1 ASCII seed encoding

ASCII encoding for a seed `*.see` file (the same as for a rule-table `*.tbl` file)<sup>18</sup> is as follows: the first byte gives the value-range  $v$ , then a byte for each value at index 0 to  $n-1$ . For example, the ASCII file, and the seed itself (in 1d), for a random seed  $v=8$ ,  $n=56$ , are shown below (note the inverted order),

*the ASCII seed file — 1+56 bytes*

```
801706771342370155163426126110422300366423711762236275130
```

*the seed itself— 56 bytes, shown numerically and graphically*

```
03157263226711732466300322401162162436155107324317760710
```



## 21.11 Saving/Loading a seed in the Golly file format

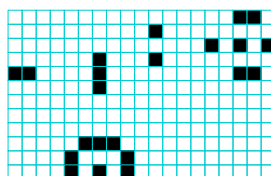
*for 2d binary networks only*

Golly is software used in the game-of-Life community (<http://www.conwaylife.com>). Golly’s pattern file format is an ascii script following “run-length” encoding, starting at the top-left of the 2d lattice, then encoding successive rows from the left. For a binary ( $v=2$ ) 2d network, numbers before the symbols ‘o’ (=1), ‘b’(=0), and ‘\$’ (=newline) stand for the number of each symbol if there are more than one, so 4\$ means finish the current row and insert three blank rows. The symbol ‘!’ terminates the script. This scrip is preceded by a preamble giving the  $x, y$  size, the rule name and possibly other information, and the total script makes up Golly’s `*.rle` file. For example, the script,

```
x = 19, y = 12, rule = x-rule-pre
```

```
16b2o$10bo$14bobobo$6bo3bo$2o4bo9b2o$6bo4$5b3o$4bo3bo$4bobobo!
```

encodes this 19×12 pattern ...



For 2d binary patterns only, DDLab can duplicate this encoding in its own `*.gly` file, but omits Golly’s preamble in the top line. DDLab does not use or need the Golly preamble because the  $x, y$  size is deduced from the ascii “run-length” script itself. In DDLab the default filename is `golly_v2.gly` (see Filing chapter 35). This file allows interchanging a seed between DDLab and Golly’s `*.rle` file, but indirectly. To convert `*.gly` to `*.rle` a Golly preamble may be added and the file renamed. To convert `*.rle` to `*.gly` the Golly preamble is removed and the file renamed.

<sup>18</sup>The ASCII rule and seed encoding (`*.tbl` and `*.see`) are the same, so the a seed can be loaded into a rule and vice-versa, provided the file-name extension is changed accordingly.

### 21.11.1 The Golly file prompt

For 2d binary ( $v=2$ ) networks, enter *Golly-G* at the seed prompt in section 21.1 to save the seed as a \*.g1y file, or to load the seed from a \*.g1y file. The following top-right prompt is presented,

**Golly ascii seed (v=2, base ij=50x60) load/save-l/s:** *(for example)*

Enter **l** to load. Both the  $i$  and  $j$  dimensions of the file lattice (deduced from the script itself) must fit within the base lattice, so base  $i \geq$  file  $i$  and base  $j \geq$  file  $j$ , otherwise a top-right error messages will appear,

**Golly: base ij=20,10 < file ij=19,12, can't load, cont-ret:** *(for example)*

If file  $ij$  is smaller than base  $ij$ , the pattern will be positioned top-left. When loading, a message similar to the following shows in the terminal,

```
load Golly: base ij=99,77 >= file ij=19,12 -OK
```

Once loaded the seed can be saved in DDLab's classic format as a \*.eed file (sections 21.4.8 and 21.8).

Enter **s** at the Golly file prompt above to save to a \*.g1y file — a message similar to the following shows in the terminal,

```
save Golly: ij=99x77 -OK
```

---



## 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, preferably in SEED-mode or TFO-mode, but also in FIELD-mode where network size is limited.

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 [16, 20], where the canalizing inputs can be tuned to move the dynamics between order and chaos (section 15). However, the Derrida plot also provides Liapunov-like insights into CA rules and rule clusters (section 22.7). New options allow automatic plots of sets of rules in ascending decimal order, filtering out equivalent binary rcode and tcode, and listing equivalence classes and rule clusters [31].

The Derrida plot, analogous to the Liapunov exponent in continuous systems, 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 normalized 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.

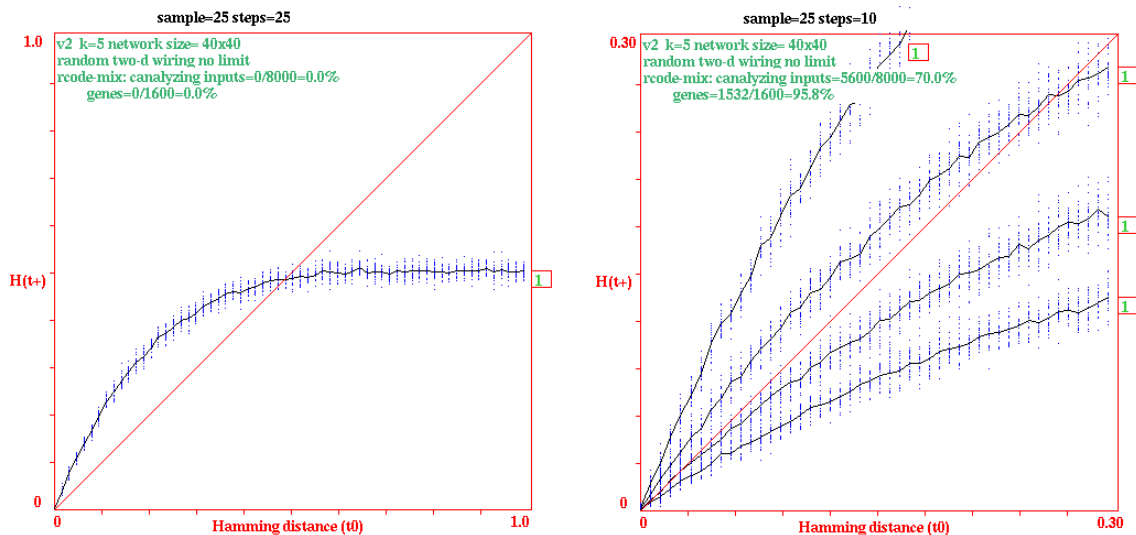


Figure 22.1: Examples of Derrida plots for a 2d random Boolean network,  $k=5$ ,  $n=40 \times 40$ . Data about the network are shown in the top-left hand corner of the graph (section 22.3), and Derrida data above the top (sample and steps). The Derrida coefficient (section 22.6) would also be shown, but is not in this case because steps  $> 1$ . The number of iterations is shown at the end of each curve.

*Left:*  $\text{Max}H=1$ , showing the full range of Hamming distance, sample=25, steps=25.

*Right:* 4 superimposed plots for increasing canalyzing settings (0%, 20%, 50% and 70%), giving progressively lower slopes.  $\text{Max}H=0.3$ , so zooming in for the most significant first part of the graph, sample=25, steps=10. Canalyzing can be changed before a new plot, and previous plots kept.

## 22.1 Selecting the Derrida plot

To select the Derrida plot, first display the network as a wiring graphic after 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*)

## 22.2 Derrida plot options

If **D** is selected in section 22.1, the following series of prompts are presented in turn in a top-right window to change the Derrida plot parameters or accept the defaults,

*for a  $40 \times 40$  RBN, values shown are examples*

**Derrida plot: quit-q, Dc sample (now 15, max 320):**

**maxH (def 1.000):** sample (def 25) tog spread (ON)-s: details-d

**iterations (def 1):** Ham steps (def 1 net=1600): keep-k:

Revised parameters generally become the new defaults. The 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 brackets are initial defaults or examples.

*parameters ... what they mean*

**quit-q** ... quit the Derrida plot.

**Dc sample (now 15, max 320):** ... the number of initial points that will form the basis of the Derrida coefficient, derived from the initial slope (section 22.6). The initial default depends on the size of the network, but is 15 for large networks.. Generally, 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),

**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  $H_{t_0} = 21$  (not normalized), the final Hamming distance, is  $H_{t_1}=16$ , 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 normalized). 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 homogeneous-*k* .  
**k-mix=k<sub>1</sub>-k<sub>2</sub>** ... the range of *k* for a *k*-mix .  
**network size=n** ... for a 1d network.  
**network size=i × j** ... for a 2d network.  
**network size=i × j × h** ... for a 3d network.  
**one-d CA wiring** ... for local 1d CA wiring.  
**two-d CA wiring** ... for local 2d CA wiring.  
**three-d CA wiring** ... for local 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: canalyzing** ... *for a rulemix*  
**inputs=c/c<sub>max</sub>=c%** ... the percentage of canalyzing inputs.  
**genes=g/n=g%** ... the percentage of canalyzing 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. The data **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** (the default) is active in section 22.2. Data at the end of each plot is described in section 22.5.2.

---

## 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 same parameters as in section 22.2 (except **Dc sample** and **maxH**) can be changed in mid-plot. The following series of prompts are presented in turn in a top-right window,

```
interrupted at Hamm=0.297, quit now-q, cont-ret:
sample (now 25):  tog spread(ON):  details-d (values shown are examples)
iterations (now 1):  Ham steps (now=1 net=1600):  keep-k:
```

Enter **q** to stop the plot at the point reached, **return** to continue the plot, or revise parameters before continuing, as in section 22.2.1. Note that changing **iterations** will have drastic effects.

---

## 22.5 Completing the Derrida plot

When the Derrida plot is complete, or if interrupted and stopped in section 22.4, data is displayed at the end of the plot (section 22.5.2), and a top-right prompt appears, with contents depending on the type rule or rulmix,

*for a rulemix*

```
Derrida plot complete
reset-r canalyzing-C:
```

*for a single binary rule (v=2) network, within the limits in table 16.2*

```
Derrida plot complete
reset-r canalyzing-C reviserule-v automatic-1/2/3 keep+k:
```

*for a single rule (v > 2) network, within the limits in table 16.2*

```
Derrida plot complete
reset-r canalyzing-C reviserule-v automatic-3 keep+k:
```

### 22.5.1 Completing the Derrida plot parameters

The meaning of the parameters in “Completing the Derrida plot” (section 22.5) are given below,

*parameters ... what they mean*

**reset-r** ... to reset the plot. The options in section 22.2 are presented, but with an additional option **redraw (no spread)-r**. Enter **r** for a quick redraw of the plot from memory without the sample spread.

**canalyzing-C** ... to revise the network’s canalyzing settings, described in chapter 15 for a rulemix, and section 18.6 for a single rule network.

- reviserule-v** ... (for a single rule network only) to review/revise the rule in a lower-right window as described in chapter 16. This is useful if assembling a number of rules on a multiple plot with option **keep+k**, or resetting the start rule for an automatic plot (22.7).
- automatic-1/2/3** ... (for a single binary rule network ( $v = 2$ ), and within the limits in table 16.2) enter **1** for automatic plots of equivalence classes [31], **2** for automatic plots of rule clusters [31] or **3** for automatic plots of rules counting up from the start rule in decimal. Options **1** and **2** also simultaneously output lists of equivalence classes or rule clusters in the terminal window (xterm) for Linux-like systems. These option are described further in section 22.7.
- automatic-3** ... (for a single rule network where  $v > 2$ , and within the limits in table 16.2) enter **3** for automatic plots of rules counting up from the start rule in decimal. These option are described further in section 22.7.
- keep+k** ... if an automatic plot was selected above, add **k** to retain previous plots for a multiple automatic plot, for example, enter **1k**, **2k** or **3k**. If **k** is not added each new automatic plot is displayed separately.

## 22.5.2 Data at the end of each plot

The small box at the end of each plot shows the following data:

- the number or iterations ... for a rulemix, or if a single rule is outside the limits in table 16.2.
- the rule in decimal ... for single rules within the limits in table 16.2, including automatic plots of sets of rules (section 22.7).
- the rule in hex ... for single rules outside the limits in table 16.2, including automatic plots of sets of rules (section 22.7).

---

## 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  $45^\circ$  ( $Dc = 0$ ) indicates the dynamics is balanced between order and chaos. A slope above the  $45^\circ$  diagonal (positive  $Dc$ ) is in the chaotic regime and correlates with increasing chaos. A slope below  $45^\circ$  (negative  $Dc$ ) is in the ordered regime and correlates with 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.

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 =  $\delta^\circ$ , then the Derrida coefficient,  $Dc = \log_2(\tan(\delta^\circ))$ . Figure 22.2 gives examples for a  $k5$  RBN, with varying degrees of canalizing (section 15) to demonstrate a range of behavior.

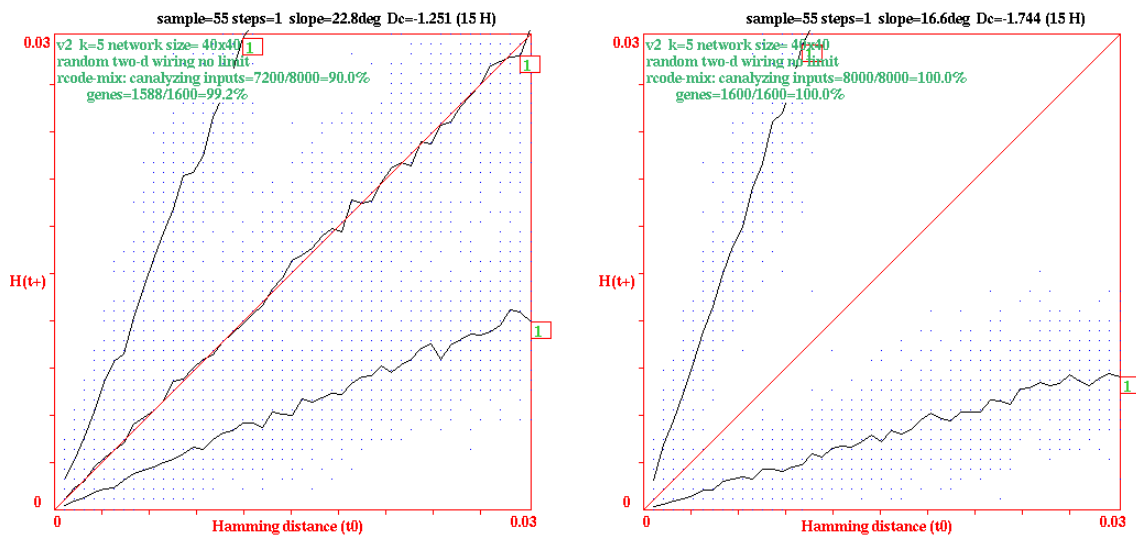


Figure 22.2: Examples of Derrida plots and their Derrida coefficients ( $D_c$ ) for a 2d RBN,  $k=5$ ,  $n=40 \times 40$  with various canalizing, settings  $C\%$ , changed in section 22.5 for a range of behavior. Other parameters were as follows:  $D_c$  sample=15,  $\text{MaxH}=0.03$ , sample=55, iterations=1, steps=1. The average slope,  $\delta^\circ$ , and the Derrida coefficient,  $D_c$ , for the plots are given below. Repeating these experiments with the same parameters will give slightly different outcomes because the exact rules in the rulemix will be differ.

$C\%$	slope $\delta^\circ$	$D_c$
0	68.2°	1.323
51	45.4°	0.019
90	22.8°	-1.251

*Left:* 3 superimposed plots:chaos/balance/order.

$C\%$	slope $\delta^\circ$	$D_c$
0	71.4°	1.572
100	16.6°	-1.744

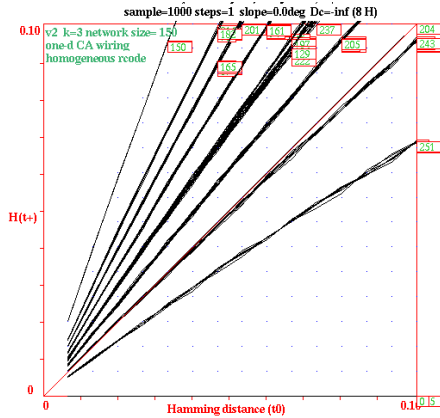
*Right:* 2 superimposed plots:extreme chaos (chain-rules) and order.

## 22.7 Automatic plots of CA rule clusters

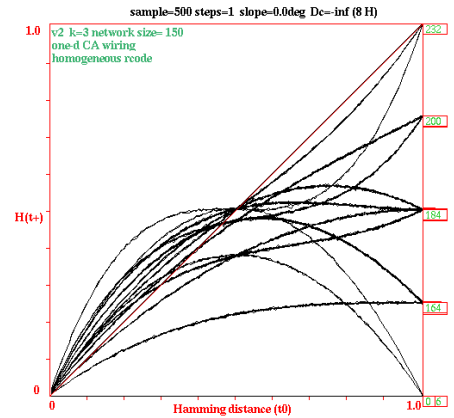
Its of some interest to compare the Derrida plots of various CA rule spaces, for example the 256 elementary rules ( $v2k3$  rocode) in figure 22.3, or the 64  $v2k5$  totalistic rules in figure 22.4 (tcode or kcode, idential for  $v=2$ ). In “The Global Dynamics of Cellular Automata” [31] it was shown that a rule has equivalent rules by negative, reflection and negative+reflection transformations, making an “equivalence class” were all behavior is strictly equivalent, originally discovered by Crayton Walker [27]. A further “complimentary” transformation will group the rules into “rule clusters”. Complimentary rules, though usually not equivalent, share some important behavior measures, including the Z-parameter and G-density [31] (section 24.9), and the in-degree histogram (section 24.6). As set out in [31], the 256 elementary rules fall into 88 equivalence classes and 48 rule clusters. The 64  $v2k5$  totalistic rules fall into 36 equivalence classes and 20 rule clusters.

```

rcode table=8 rulespace=256
#: s n r nr Dc
1: 0 255 0 255 -inf
2: 1 127 1 127 -0.442
3: 2 191 16 247 -0.430
4: 3 63 17 119 -0.025
5: 4 223 4 223 -0.431
6: 5 95 5 95 -0.046
7: 6 159 20 215 0.532
8: 7 31 21 87 0.307
9: 8 239 64 253 -0.459
10: 9 111 65 125 0.529
11: 10 175 80 245 -0.023
12: 11 47 81 117 0.302
13: 12 207 68 221 -0.020
14: 13 79 69 93 0.309
15: 14 143 84 213 0.289
16: 15 15 85 85 0.000
17: 18 183 18 183 0.533
18: 19 55 19 55 0.308
19: 22 151 22 151 1.112
20: 23 23 23 23 0.556
21: 24 231 66 189 0.551
22: 25 103 67 61 0.773
23: 26 167 82 181 0.772
24: 27 39 83 53 0.556
25: 28 199 70 157 0.779
26: 29 71 71 29 0.571
27: 30 135 86 149 0.960
28: 32 251 32 251 -0.431
29: 33 123 33 123 0.541
30: 34 187 48 243 -0.012
31: 35 59 49 115 0.284
32: 36 219 36 219 0.543
33: 37 91 37 91 0.776
34: 38 155 52 211 0.773
35: 40 235 96 249 0.547
36: 41 107 97 121 1.121
37: 42 171 112 241 0.305
38: 43 43 113 113 0.545
39: 44 203 100 217 0.771
40: 45 75 101 89 0.953
41: 46 139 116 209 0.555
42: 50 179 50 179 0.304
43: 51 51 51 51 0.000
44: 54 147 54 147 0.955
45: 56 227 98 185 0.767
46: 57 99 99 57 0.956
47: 58 163 114 177 0.564
48: 60 195 102 153 0.964
49: 62 131 118 145 0.771
50: 72 237 72 237 0.544
51: 73 109 73 109 1.114
52: 74 173 88 229 0.770
53: 76 205 76 205 0.294
54: 77 77 77 77 0.548
55: 78 141 92 197 0.559
56: 90 165 90 165 0.968
57: 94 133 94 133 0.768
58: 104 233 104 233 1.118
59: 105 105 105 105 1.512
60: 106 169 120 225 0.959
61: 108 201 108 201 0.959
62: 110 137 124 193 0.785
63: 122 161 122 161 0.763
64: 126 129 126 129 0.540
65: 128 254 128 254 -0.418
66: 130 190 144 246 0.542
67: 132 222 132 222 0.545
68: 134 158 148 214 1.117
69: 136 238 192 252 -0.034
70: 138 174 208 244 0.316
71: 140 206 196 220 0.295
72: 142 142 212 212 0.554
73: 146 182 146 182 1.110
74: 150 150 150 150 1.515
75: 152 230 194 188 0.777
76: 154 166 210 180 0.950
77: 156 198 198 156 0.961
78: 160 250 160 250 -0.035
79: 162 186 176 242 0.295
80: 164 218 164 218 0.764
81: 168 234 224 248 0.304
82: 170 170 240 240 0.000
83: 172 202 228 216 0.561
84: 178 178 178 178 0.558
85: 184 226 226 184 0.553
86: 200 236 200 236 0.291
87: 204 204 204 204 0.000
88: 232 232 232 232 0.555
    
```



MaxH=0.1, enough to compute Dc



MaxH=1, the full range.

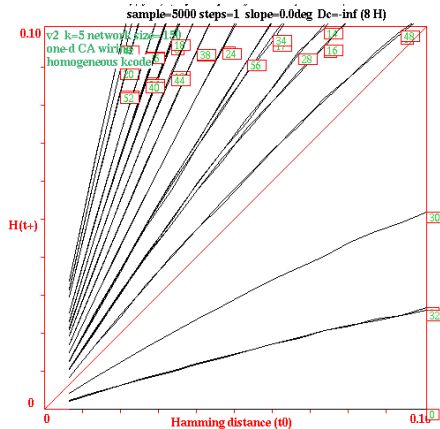
Figure 22.3: Automatically generated Derrida plots of the 256 elementary rules ( $v2k3$ ). *Left*: the printout in the terminal (xterm) of the 88 equivalence classes. The headings #: s n r nr Dc refer to #=count: s=start, n=negative, r=reflection, nr=negative+reflection, and Dc=Derrida coefficient. *Above Left*: the corresponding Derrida plots for MaxH=0.1, enough to compute Dc.

*Above Right*: Derrida plots for the full range of Hamming distance, MaxH=1, for the 48 rule clusters. In this case an example of the terminal printout is as follows:

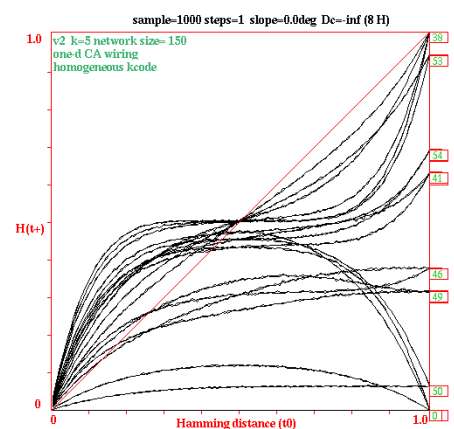
```

42: 62 131 118 145 0.794 (equivalence class, and Dc for cluster 42 with start rcode 0)
193 124 137 110 0.797 (-0.003) (complimentary equivalence class, Dc, Dc difference)
    
```

Each rule cluster produces the same Derrida plot and Dc. However, it turns out that just 14 Derrida plots cover the whole of rulespace.



MaxH=0.1, enough to compute Dc



MaxH=1, the full range.

Figure 22.4: Analogous plots to figure 22.3 of the 64  $v2k5$  tcodes. The terminal printout will be just #: s n Dc because tcode and kcode rules are symmetric, so reflections reproduce the same rule. *Above Left*: for 36 equivalence classes, MaxH=0.1, enough to compute Dc (Dc sample=8H). *Above Right*: for 20 rule clusters, for the full range of Hamming distance



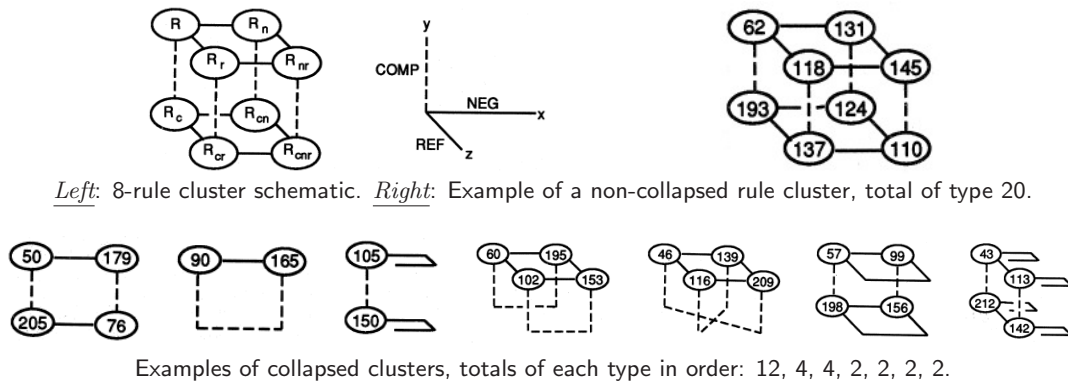


Figure 22.5: Graphical representation of rule clusters of the  $v2k3$  “elementary” rules, and examples, taken from “The Global Dynamics of Cellular Automata” [31] section 3.3, where it is shown that the 256 rule-space breaks down into 88 equivalence classes and 48 clusters. The rule cluster is depicted as 2 complimentary sets of 4 equivalent rules at the corners of a box — with negative, reflection, and complimentary links on the  $x, y, z$  edges, but the edges also collapse due to symmetries.

Experiment confirms, as expected, that the Derrida plots for the rules in an equivalence class will be the same (subject to sampling variation) — as if plotting the same rule, and this turns out also to be true of the complimentary rules — the rule cluster has the same Derrida properties (figure 22.4). In DDLab it is now possible to automatically generate the Derrida plots for binary ( $v=2$ ) rule-spaces according to these equivalence classes or rule clusters, or for ( $v \geq 2$ ) rules, counting up from the start rule in decimal. The procedure works for rcode in SEED-mode, and for kcode or tcode in TFO-mode. However, rule-tables must be within the limits in table 16.2 for expressing the rule as a decimal number.

To automatically generate the Derrida plots, in section 22.5 enter **1** for equivalence classes, **2** for rule clusters (figures 22.3 and 22.4), or **3** for just a sequence of rules. In each case the initial rule, selected in section 16 or revised with **reset-r** in section 22.5 starts of the sequence, counting up, so to obtain an orderly and complete sequence of rules for an entire rule-space, the initial rule should be set to decimal rule 0 (zero).

Starting with rule 0 for equivalence classes, just the lowest decimal rule will be plotted (as the representative rule) and its equivalents skipped and filtered from the remaining list. For rule clusters, the compliment of the representative rule is also plotted, and its equivalents skipped and filtered. The rules making up equivalence classes or rule clusters are displayed in the terminal (xterm) for Linux-like systems (figure 22.3), so this is a useful result in itself — Derrida parameters can be minimised if equivalence classes are the only data required.

## Chapter 23

# Graphic conventions for attractor basins

*not in TFO-mode*

Figure 23.1 and section 23.1 explain 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.

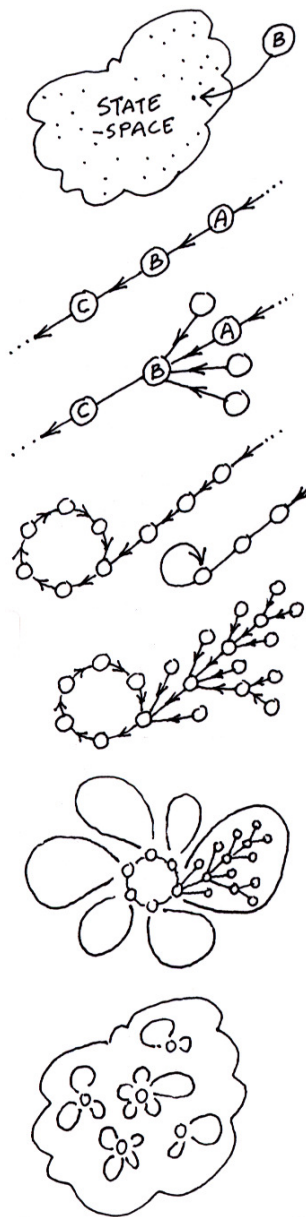
This chapter describes the graphic conventions for drawing the state transition graphs of attractor basins and subtrees, illustrated in figures 23.2 — 23.5, and in other figures in the manual.

---

### 23.1 Basins of Attraction — the idea

Given an invariant network architecture and the absence of noise, the dynamics on a discrete dynamical network 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.



For a binary network size  $n$ , an example of one of its states  $B$  might be 1010...0110. *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: The idea of basins of attraction in discrete dynamical networks (section 23.1).

## 23.2 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 26.3).

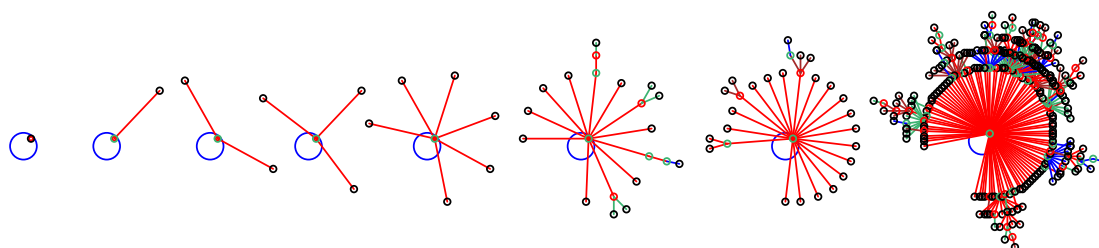


Figure 23.2: A point attractor (period=1) is represented as a node cycling to itself. The single node and the center line of the pre-image fan (level 1) are set at a default angle of  $45^\circ$ . The examples above show point attractors with increasing fan size, from 0 upwards (from  $v2k3$  rcode(dec) 77,  $n=12$ ).

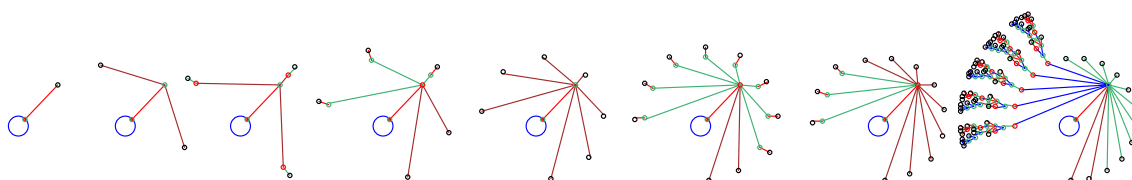


Figure 23.3: The pre-image fan (level 2) of a single pre-image (level 1) of a point attractor. The examples show increasing fan sizes, from 0 upwards (from  $v2k3$  rcode(dec) 110,  $n=(1-3)$  and  $n=(6-10)$ ). In this case both the point attractor and its pre-image are uniform states (all0s and all1s) so the pre-image fan and attached trees are organized into equivalent groups (section 23.4.2).

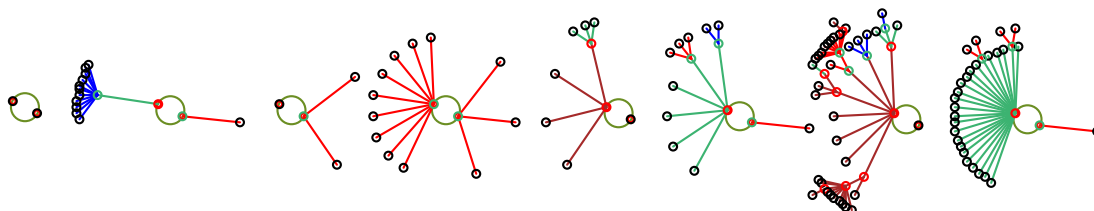


Figure 23.4: Period 2 attractors which cycle to each other. The two nodes are set at a default angle of  $-15^\circ$ . Varying sizes of the pre-image fan to both attractor states are shown, from 0 upwards. A fan at level 1 is slightly angled to indicate that the direction of time is clockwise in the attractor — the center lines of subsequent fans are radial to the center of the attractor cycle (taken from rcode  $v2k3$  rcode(dec) 33,  $n=13$ ).

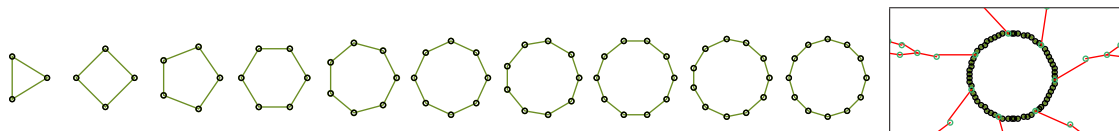


Figure 23.5: Attractors with periods  $\geq 3$  are represented by polygons whose diameter approaches an upper limit with increasing period. The first 10 examples from the left have periods of 3 to 12 (from rcode(dec) 248,  $n=(3-12)$ ). The framed example on the right is a fragment of a basin from  $v2k3$  rcode(dec) 30,  $n=14$ , with 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 center lines of subsequent fans are radial to the center of the attractor cycle — see figure 2.6 for a clearer example.

---

## 23.3 Attractor cycles

Where three 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.

---

## 23.4 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 center line of the fan angle is radial to the attractor cycle center 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 2.6).

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, but both the rate of decrease and the number of levels can be reset (section 25.2.2).

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* (section 26.2), equivalent trees will be computed and drawn simultaneously).

### 23.4.1 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 26.4.3).

The edges in the same pre-image fan are drawn in the same color (except for the uniform states — section 23.4.2 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 26.2) equivalent trees will be colored identically. Conventions for bit pattern node colors are described in section 26.3.1.

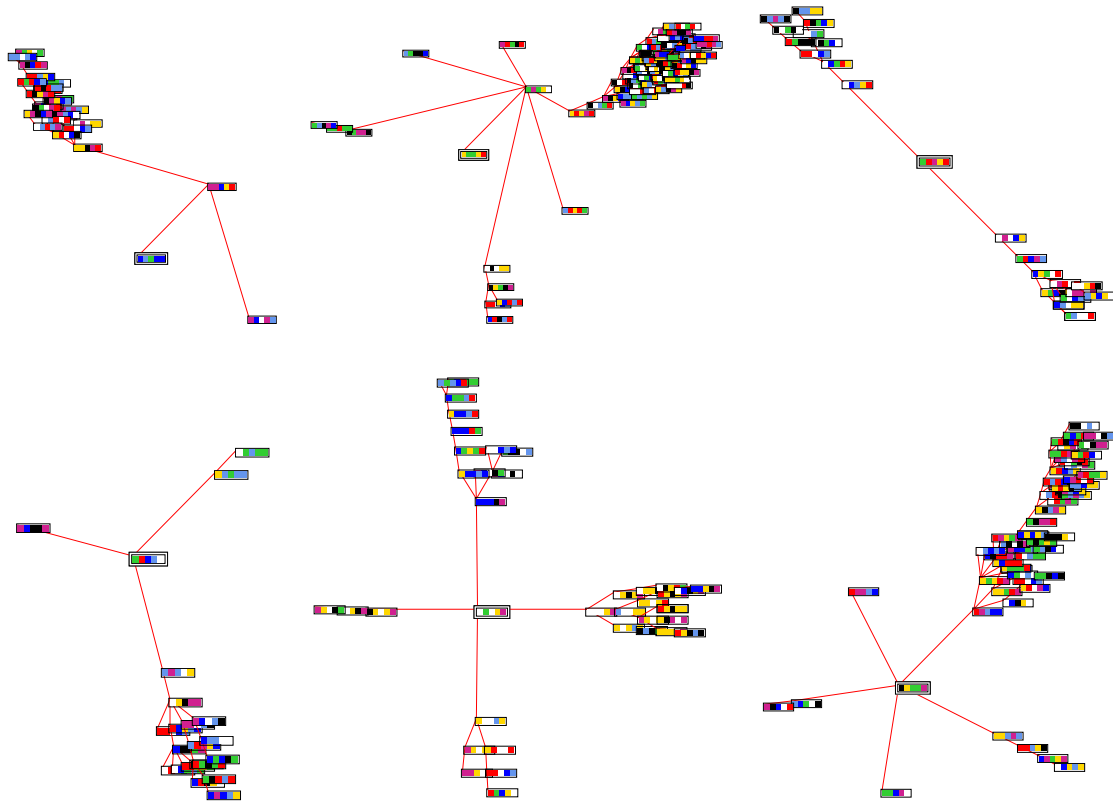


Figure 23.6: Examples of subtrees from root states with in-degrees (at level 1) of 1 to 5. These examples are for  $v8k3$  CA,  $n=5$  for various rules, and random initial states, running forward by about 3 time-steps before running backward. States are shown as value patterns — the root state is highlighted by default inside a double outline. For in-degree 1 at level 1 (the two subtrees top-left) similar layout principles apply as in figure 23.3.

### 23.4.2 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 the same 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 [31] 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 26.5).

### 23.4.3 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. Running forward before running backward is usually necessary because most states have no pre-images — they are leaves, 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.

---

## Chapter 24

# Output parameters for attractor basins

*not in TFO-mode.*

There are many 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 options have defaults so the remainder can be skipped at any time – [accept defaults-d](#).

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/boundary-p** ... display of attractor basins and nodes (chapter 26), and null boundary conditions (NBC, section 26.1).
- data/pause-t** ... pausing between trees/basins, specifying data to be recorded (chapter 27).
- PScript-P** ... to create a vector PostScript file of the attractor basin/s (section 24.2).
- jump-j** ... (*FIELD-mode only*) to display the jump-graph (section 24.3).
- 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 [basin 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/boundary-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/boundary-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 section 24.1 are summarized below, and described in detail in this chapter and chapters 25 to 28.

As soon as a “**revise from**” option is selected, the reminder **accept defaults-d** appears in a top-center window – enter **d** (at any time) to accept all remaining defaults, and skip further output parameter prompts.

#### *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 — described in 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/boundary-p** ... to set null boundary conditions (NBC, section 26.1) instead of periodic (PBC), to disable the compression of equivalent basins and trees (the default for CA, section 26.2), and for a sequence of display prompts, 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 (section 24.2).

**jump-j** ... (*FIELD-mode only*) to activate the jump-graph (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.

---

## 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 35.3). 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<sup>1</sup> 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 (with edges), or **jL** for “layout only” — where the jump-graph methods are applied without edges for just laying out attractors in any arbitrary position.

If the genuine jump-graph (with edges) is selected, “compression” for CA will be turned off automatically, with the following message,

... - **compression OFF:** (*if compression on, enter return to continue*)

The jump-graph (section 20.3) 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 each 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.

---

<sup>1</sup>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/deactivated 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 fitness in the “density classification problem” [12, 18] (section 24.7).
- screensave demo** ... (*SEED-mode only*) a continuous demo of single basins or subtrees for different or mutant rules (sections 4.11, 24.8).
- G-density, Z and  $\lambda$**  ... 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 24.10) which can also be activated/deactivated on-the-fly (section 30.3).
- speed** ... to slow down drawing attractor basins and backwards space-time patterns or revert to full speed. See section 24.11, which also lists other stages in DDLab where slow motion can be invoked, including on-the-fly.

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).

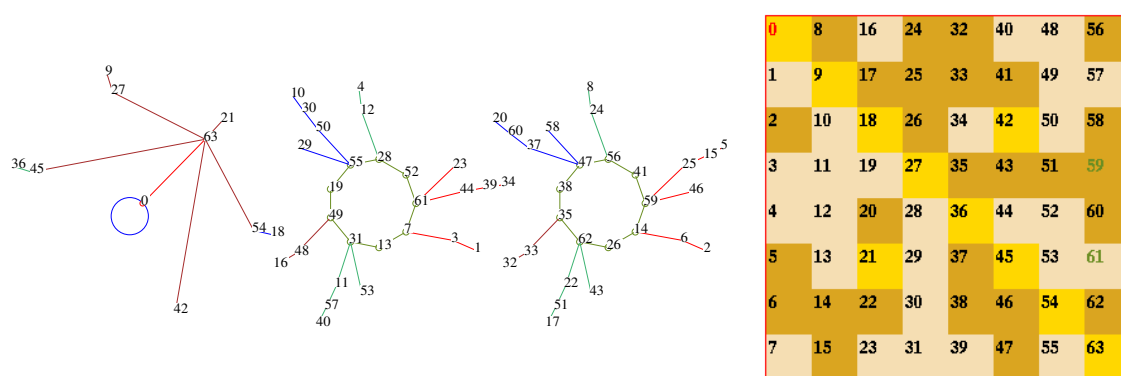


Figure 24.1: The state-space matrix of the basin of attraction field of a cellular automaton, *v2k3* rcode (dec)110,  $n=6$ . *Left*: the uncompressed state transition graph. decimal numbers representing states. *Right*: The state-space matrix, where the 3 basins are represented by 3 colors. The decimal 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 summary

*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:**

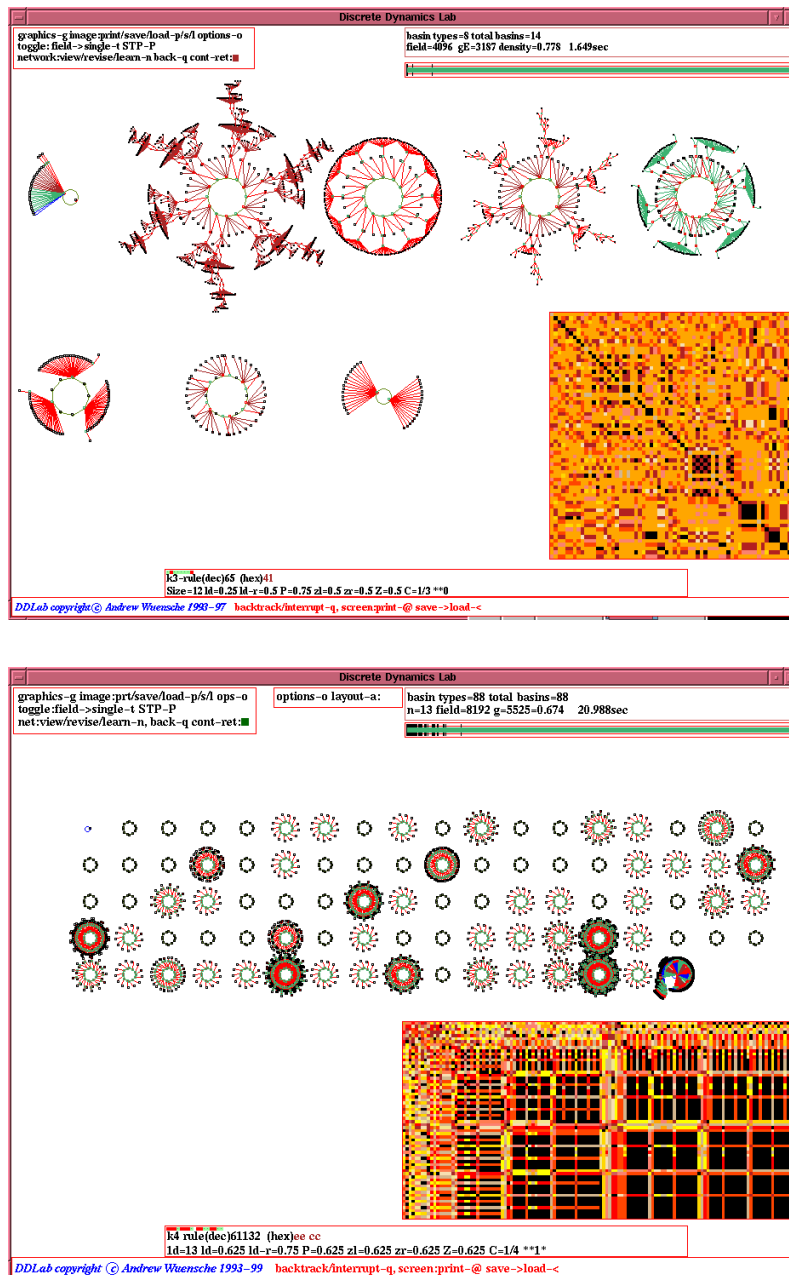


Figure 24.2: The state-space matrix (lower right on each screen) of the basin of attraction field of a CA, with compression off. *Top:*  $v2k3$  rcode (dec)65,  $n=12$ , the 8 basins types are represented by 8 colors in the matrix. *Bottom:*  $v2k4$  rcode (hex)eecc,  $n=13$ . Because  $n$  is odd, and the extra bit is included on the left, the grid is a flat rectangle. The 88 basins are represented by cycling through 15 colors.

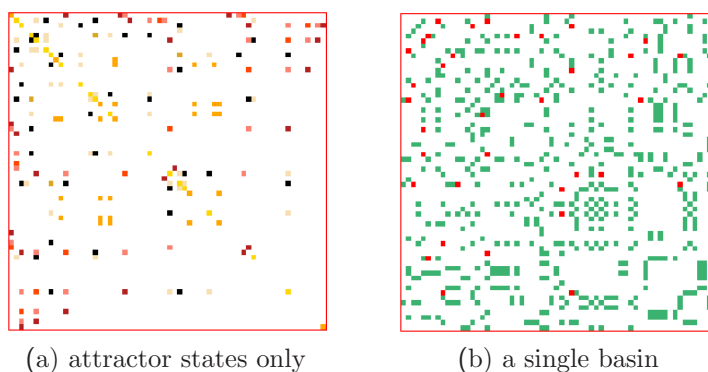


Figure 24.3: The state-space matrix of the same CA in figure 24.2 *top*, *v2k3* rcode (dec)65,  $n=12$ .  
 (a) just the attractor states of the of the 8 basin types represented by 8 colors.  
 (b) just the states in the 5th basin type (top-right in figure 24.2 *top*). The attractor and transient states 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 appears in the bottom title bar (section 5.5),

```
matrix-m STP:tog/scroll/esp/contr-s/#/e/c slow/max:</>
```

Enter **m** to toggle the state-space matrix on-the-fly. This can also be done for space-time patterns (section 31.2.2.1). Options **s/#/e/c** and **</>** are described in section (section 30.3).

## 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 top-right prompt is presented in section 24.4,

```
in-degree 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 (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 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):
```

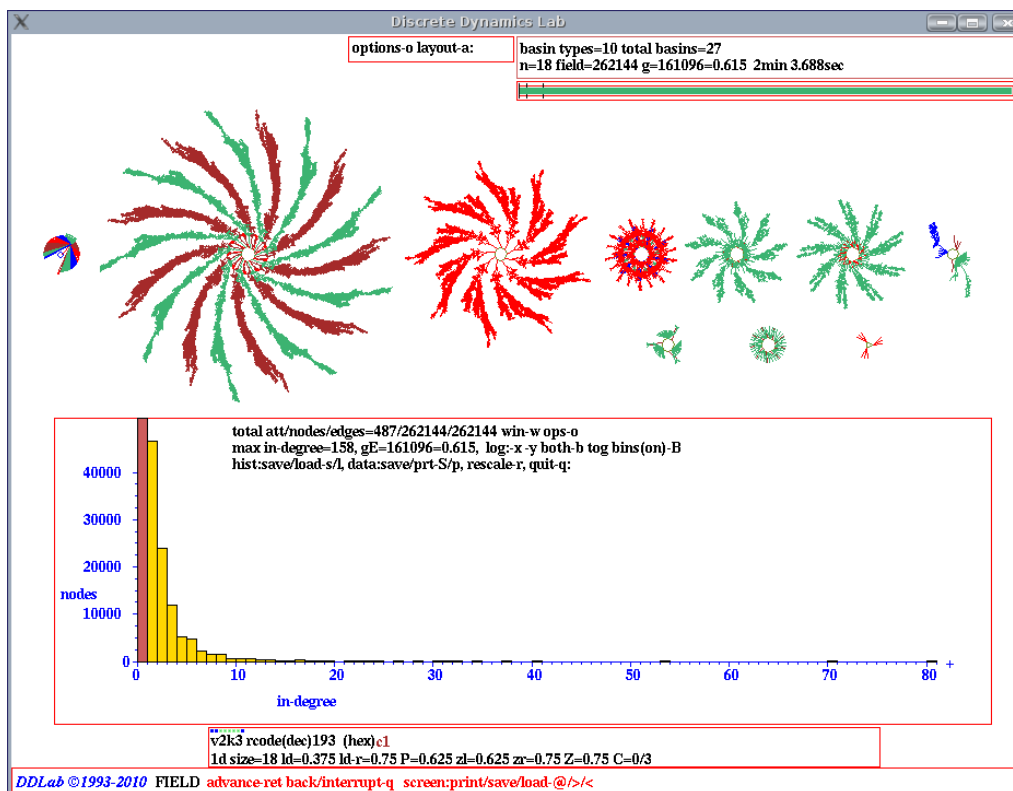


Figure 24.4: In-degree histogram which has been rescaled from the default settings, of a basin of attraction field of a CA,  $v2k3$ , rcode (dec)193,  $n=18$ . The attractor basins are shown at the top.

## 24.6.2 Drawing the in-degree histogram

While the in-degree histogram is active, Attractor basins are drawn in the usual way, and in-degree information is recorded. The histogram and data will be displayed in a window in the lower section of the screen once the attractor basin is complete, or can be seen at any time if paused on-the-fly (enter **q**), in which case a top-right prompt similar to the following is presented (variations and other pause options are described in section 30.2),

**early exit - in pre-image fan** (*if within a pre-image fan*)  
**inhist-h, next-tree/basin-n options-o stopfield-q speed-V cont-ret:** (*for a basin field*)

During a pause, enter **h** to see the histogram and data corresponding to the attractor 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. The  $x$  and  $y$  axis are automatically re-scaled according to the range of in-degrees encountered, but can also be manually re-scaled or shown in log form for a clearer view of the spread of frequencies or to amplify smaller in-degrees.

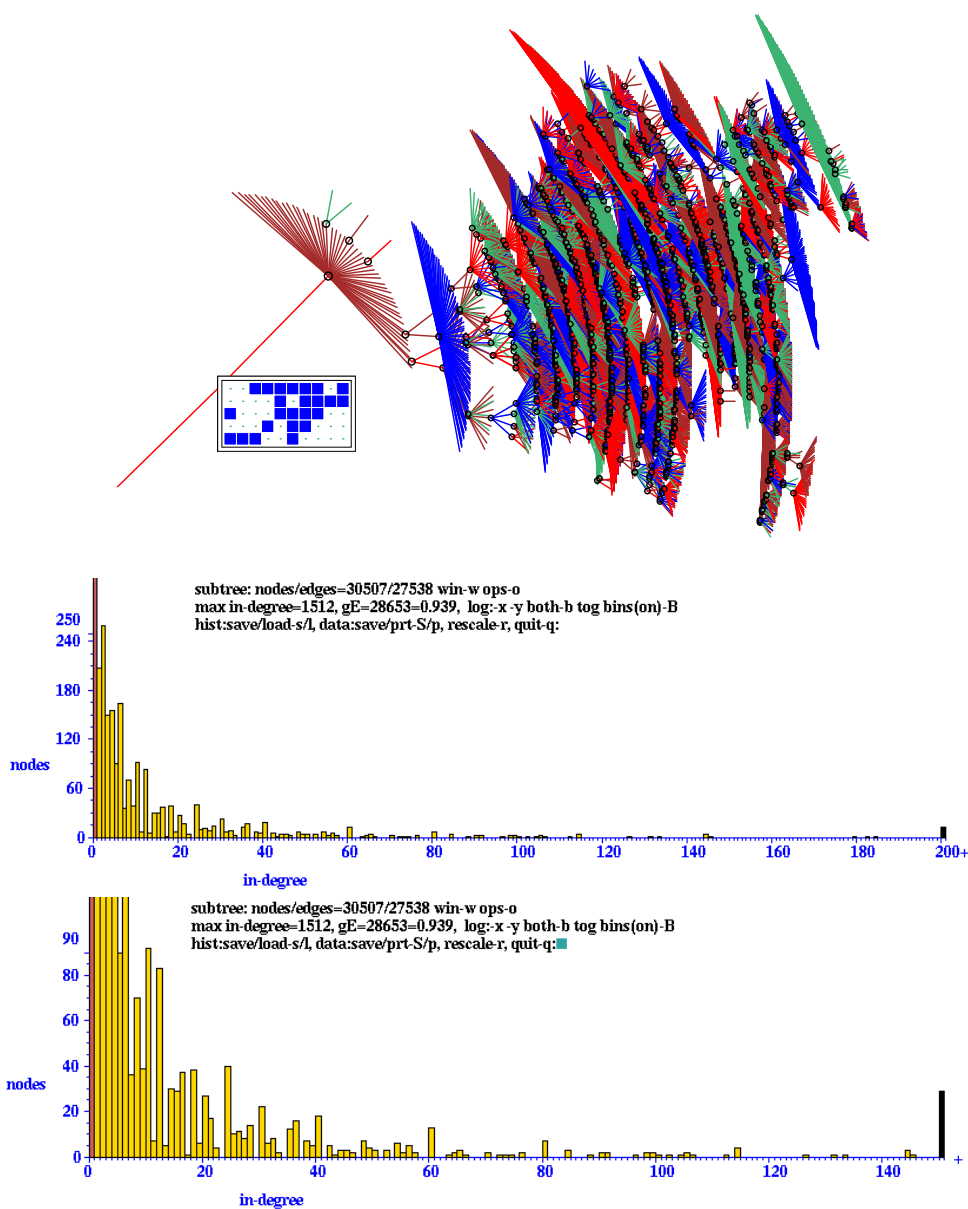


Figure 24.5: *Top*: A subtree of a CA,  $v2k3$  rcode (dec)193, with the  $n=50$  1d root state shown in 2d  $10 \times 5$ , leaf nodes suppressed and the STG rotated by  $90^\circ$ . *Below*: Two versions of the in-degree histogram, with increasing rescaling ( $x$  and  $y$  axis) for a closer view. 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.



### 24.6.3 In-degree data and prompts

An example of the data displayed in the window, and initial prompts (figure 24.4) for the basin of attraction field of a  $v2k3$  CA,  $\text{rcode}(\text{dec})193$ ,  $n=18$ ) 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 prompt starts with **subtree: nodes/edges= ...**

### 24.6.4 in-degree window — data decode

The data shown in the in-degree frequency window (section 24.6.2) signifies the following,

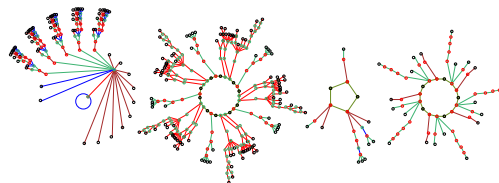
```
total att ... the number of states on attractor cycles (does not apply to subtrees).
nodes ... the number of nodes whose pre-images have been computed.
edges ... the total number of edges computed so far.
max in-degree ... the largest in-degree found, possibly more than the cut-off limit.
gE ... the frequency and fraction of garden-of-Eden states (in-degree 0), usually
the greatest frequency.
```

Note that for a complete basin (or field),  $\text{nodes}=\text{edges}$ , for a complete subtree,  $\text{nodes}=\text{edges}+1$ , and for an incomplete attractor basin, (i.e. if interrupted),  $\text{nodes}<\text{edges}$ .

### 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 data example is for CA,  $v2k3$  rule 193,  $n=10$ 
(basin of attraction field inset).
```

```
pre-image frequency distribution
0: 442
1: 350
2: 136
3: 45
4: 30
6: 10
7: 10
17: 1
```



```
rescale-r ... to rescale the histogram (section 24.6.7).
```

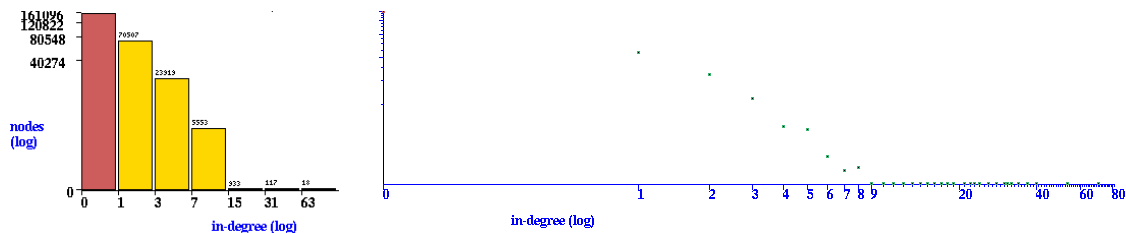


Figure 24.6: The in-degree histogram as a log-log plot can be implemented in two ways on the  $x$ -axis, *Left*: preserving a histogram (the default) with bins that capture an increasing range of in-degrees, 0, 1–2, 3–7, 8–15, etc. *Right*: a simple log-log plot. The  $x$  and  $y$  axis can also be treated separately. These plots are for the `CA, v2k3 rcode (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  $\leq 30$*

**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 to categorize states between a majority 0s and 1s. In this example of typical space-time patterns ( $n=149$ ), *v2k7* rcode (hex)(ff f0 8c f0 ff e0 00 f0 ff f0 00 f0 ef e0 00 e0), evolved by Raja Das [12].

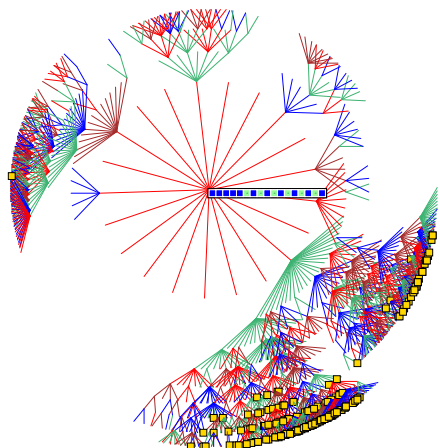


Figure 24.8: A subtree showing exceptions to density classification. Using the Raja Das [12] rule (as in figure 24.7), a subtree was generated from an initial state with 11 1's out of  $n=17$ . Most states in the subtree also have a majority of 1s, but there are 9.63% exceptions, indicated in the “Data on subtrees” window (section 27.2.6) shown below, and by yellow squares in the subtree.

```
subtree=1901 maj0=183=0.0963 root(hex)=01 f5 55
g=1510=0.794 ml=9 mp=26
```

## 24.7 Density classification problem — attractor basins

The “density classification problem” is a favourite exercise in emergent computation [12, 18] 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 (implemented in section 31.7.1).

This section introduces 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 (figures 24.7, 24.9). Likewise, a subtree with a given majority should attract the same majority value (figure 24.8). 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 top-right prompt is presented in section 24.4,

**show/count states majority, enter (0 - 1):** (*for v=2*)

Enter **1** or **0** to highlight the majority states, which are displayed as small yellow squares with a black border (figure 24.8). The number and percentage of the value selected is also displayed in the data window when a basin or subtree is complete (sections 27.2.3, 27.2.6) as in figure 24.8.

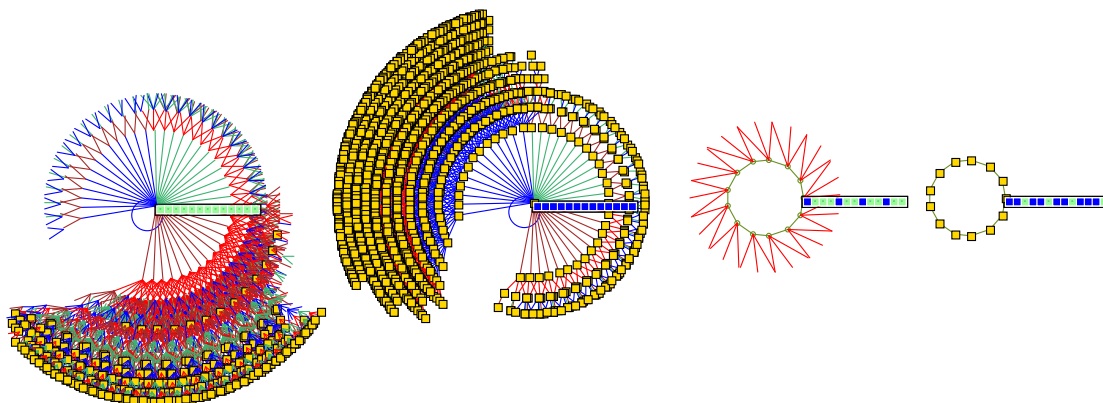


Figure 24.9: A basin of attraction field showing states with a majority of 1s as yellow squares.  $n=13$ ,  $v2k7$ , same rule as in figure 24.8. Nearly all states belong to the all 0's and all 1's point attractors. Data on the relative volume of these basins and (density of majority-1s) within each: 51.9% (7.34%) and 47.3% (97.3%) — taken from the basin data window (section 27.2.3) at a pause (section 27.1.2).

## 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 top-right 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, 24.10). Most setting, (network, layout, display, mutation) apply to the screen-saver demo. For a quick-start example see section 4.11.

## 24.9 $G$ -density, $Z$ and $\lambda$

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 [38]. A number of static parameters from the rcode-table itself also predict convergence, to varying degrees. Although these parameters are generalized for multi-value  $v \geq 3$ , the options involving parameters (sections 24.9.3 – 24.9.5) apply only to binary ( $v=2$ ) rules. If  $S$  is the rcode size, the binary parameters can be defined as follows,

- $\lambda$ -parameter ... the fraction of 1s in  $S$ . For  $v \geq 3$   $\lambda$  is the fraction of non-quiescent values [22].
- $\lambda_{ratio}$  ... the total of the lesser value (1s or 0s) divided by  $S/2$ .
- $Z$ -parameter ... the probability that the next unknown cell in the CA reverse algorithm is determined [31, 38, 48].

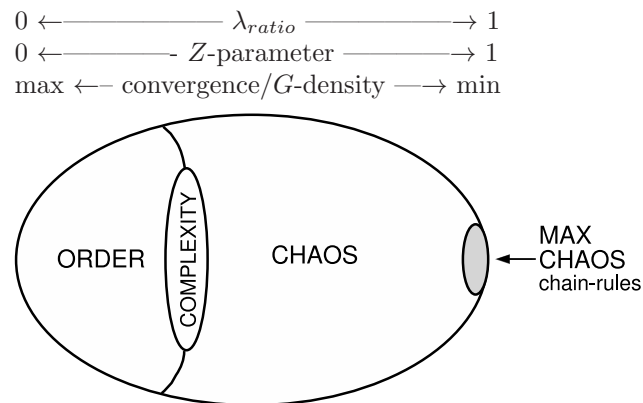


Figure 24.10: A view of rule-space (after Langton [22] with max chaos added). Tuning the  $Z$ -parameter from 0 to 1 shifts the dynamics from maximum to minimum convergence, from order to chaos, traversing a phase transition where complexity lurks. The chain-rules (about the square root of rule-space) 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  $\lambda_{ratio}$  — only  $Z$  is able to identify the maximally chaotic chain-rules (section 16.11). Complex rules occur at the order/chaos phase transition, but are difficult to identify by these static parameters<sup>2</sup>.

$\lambda_{ratio}$  is utilised instead of the  $\lambda$ -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.  $\lambda_{ratio}$  approximates  $Z$ , which depends on the distribution as well as the proportions of values in the rcode —  $Z$  predicts behavior more accurately (figure 24.13). To test these relationships, this section describes how graphs of  $G$ -density against  $Z$  and/or  $\lambda_{ratio}$  can be plotted for predefined CA rules.

Plots can be made of  $Z$  against  $\lambda_{ratio}$  (and  $Z_{left}$  against  $Z_{right}$ ) to relate parameters for predefined CA rule samples. The proportions of rules in rule-space with various levels of canalizing inputs (chapter 15),  $\lambda_{ratio}$ , and the  $P$ -parameter<sup>3</sup> may be computed and displayed (figures 24.1, 24.2) — these measures are of interest in predicting convergence and stability in random Boolean networks.

Plots of the variation of  $G$ -density with network size are also available, another useful measure of the order-chaos spectrum of behavior applied mainly to CA [33, 47] (figure 24.12).

The following top-right prompt is presented in section 24.4,

```
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 ≥ 3 the G-density graph only
G-density plot: size range-3
```

<sup>2</sup>Chapter 33 describes alternative methods for automatically classifying rule-space including complex rules.

<sup>3</sup>The  $P$ -parameter [16] is the fraction of 0s or 1s in the rule-table (whichever is more),  $P$  varies from 1 (order) to .5 (chaos). The  $P$ -parameter is favoured in theoretical biology over the equivalent  $\lambda$ -parameter or  $\lambda_{ratio}$ .

### 24.9.1 $G$ -density, $Z$ and $\lambda$ — 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 24.9.3 – 24.9.3.2).
- +Lambda ratio-2** ... enter **2** for two simultaneous plots,  $Z$  against  $G$ -density, and  $\lambda_{ratio}$  against  $G$ -density (sections 24.9.3 – 24.9.3.2).
- size range-3** ... enter **3** to plot  $G$ -density against network size  $n$ . (section 24.9.2).
- parameter graph :** ... *plots between parameters* enter **L** for a predefined rule sample ...
- Zpara-Lda** ... to plot  $Z$  against  $\lambda_{ratio}$  (sections 24.9.4 – 24.9.4.3).
- Zleft-Zright** ... to plot  $Z_{left}$  against  $Z_{right}$  (sections 24.9.4 – 24.9.4.3).  
The  $Z$ -parameter is the greater of the two.
- (C-P data)** ... to see the proportions of rules in rule-space —  $\lambda_{ratio}$ ,  $P$ -parameter, and canalizing inputs (section 24.9.5). This can be activated once the  $Z$  —  $\lambda_{ratio}$  or  $Z_{left}$  against  $Z_{right}$  plots (above) are complete.

### 24.9.2 $G$ -density plotted against network size

for to multi-value,  $v \geq 3$ , as well as binary  $v=2$

The rate of increase of  $G$ -density with network size (system size)  $n$  is another order-chaos indicator in CA. For the vast majority of rules,  $G$ -density increases with  $n$ , with ordered dynamics showing the highest rate [38]. Very chaotic rules may approximate a zero rate, but only the maximally chaotic chain-rules show  $G$ -density decreasing with increasing  $n$  (figure 24.12).

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:**  
**start size (def=5):**      **end size (max=31):**      (*end size < 20 is recommended*)

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 (chapter 28). The plot will be drawn in the lower right of the screen. The basin of attraction fields (or single basins, or subtrees) will be draw (at a small scale by default) in a window in the upper part of the screen (figure 24.11). When each attractor basin is complete the  $G$ -density ( $y$ -axis) is plotted against the network size  $n$ , and the plot will continue automatically for successive values of  $n$  (figure 24.12). Once complete, the following prompt appears in the top-right corner of the data window (section 27.2),

... **quit-q rule-r mut-(def) keep+k:**

Enter **r** to select a new rule (set in a lower-right window, chapter 16). To superimpose a new plot on an old plot enter or add **k**, – i.e. enter **rk** for a new rule, or just **k** for the next 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 plot and backtrack.

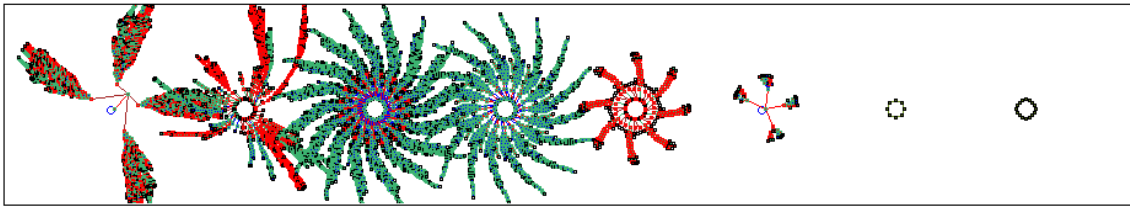


Figure 24.11: As the plot of  $G$ -density against network size  $n$  is in progress, and also the plots against  $\lambda_{ratio}$  or  $Z$  (figure 24.13), the attractor basins are drawn in an upper window at a default scale and layout (which can be changed in chapter 25). This example shows the basin of attraction field for 1d CA,  $v2k5$  rcode 9569a999,  $n=16$ , in figure 24.12.

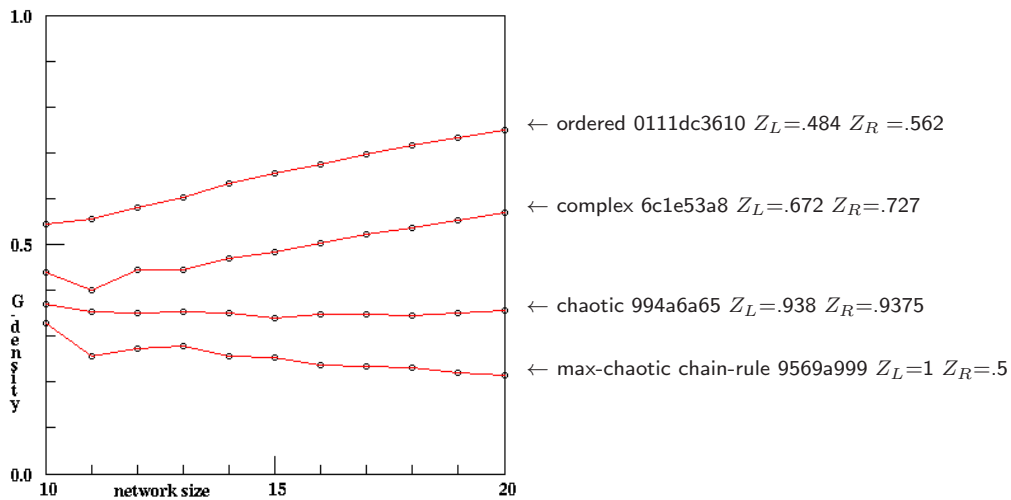


Figure 24.12: Plotting  $G$ -density against a range of  $n$  for ordered, complex and chaotic 1d CA,  $v2k5$  (rcode indicated), where the ordered rule has the highest rate of increase of  $G$ -density. Very chaotic rules may approximate a zero rate; only chain-rules show  $G$ -density decreasing.

### 24.9.3 $G$ -density against $Z$ and/or $\lambda_{ratio}$ for binary, $v=2$ , rcode or tcode only — not for kcode

Enter **1** in section 24.9 for a graph of  $G$ -density against  $Z$ , or **2** for an additional graph of  $G$ -density against  $\lambda_{ratio}$  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  $\lambda_{ratio}$  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 24.11. When each attractor basin is complete the  $G$ -density ( $y$ -axis) is plotted against the rule's  $Z$  and/or  $\lambda_{ratio}$  on the  $x$ -axis.

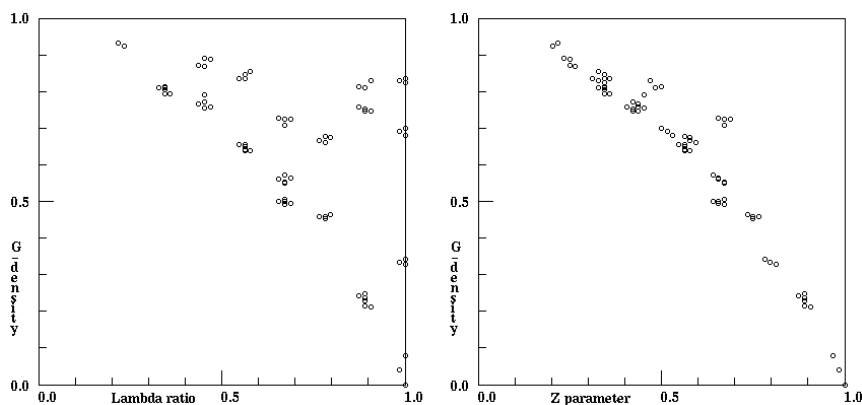


Figure 24.13:  $G$ -density plotted against *Left*:  $\lambda_{ratio}$ , and *Right*: the  $Z$  parameter.  $n=12$ , for the set of all  $v2k7$  totalistic rules (tcode). The complete basin of attraction field was generated for each tcode and the garden-of-Eden states counted.  $Z$  provides a tighter prediction of  $G$ -density than  $\lambda_{ratio}$ .

### 24.9.3.1 $G$ -density for tcode, or rcode $k \leq 3$

For tcode, or rcode if  $k \leq 3$ , the rules included in the plot consist of a countdown from the currently active rcode or tcode as a decimal number. To include the entire rule-space, the maximum rule should be selected. An easy way to do this is to enter *fill-f* in section 16.1.1, which sets all bits to 1.

The range of  $Z$  for the plot may be restricted, with a lower and upper limit. Note that low values of  $Z$  (especially  $Z=0$ ) should be avoided except for small  $n$ , 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):**

### 24.9.3.2 $G$ -density for $k > 3$ rcode

For rcode where  $k > 3$ , 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  $Z$ .

## 24.9.4 $Z - \lambda_{ratio}$ plots

*for binary,  $v=2$ , rcode or tcode only — not for kcode*

Enter **L** in section 24.9 to plot of the  $Z$ -parameter ( $x$ -axis) against  $\lambda_{ratio}$ , 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  $\lambda_{ratio}$ , the  $P$  parameter, and analyzing inputs may be displayed once the graph is complete (section 24.9.5). The measures are computed directly from rule-tables. The following prompt is presented,

**plot: Zleft-Zright-1, Zpara - Lda ratio and C-P data-(def):**



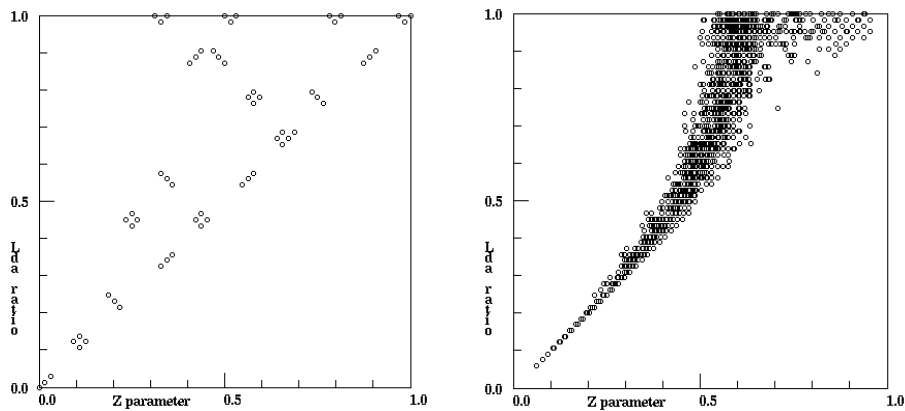


Figure 24.14:  $\lambda_{ratio}$  plotted against  $Z$ . *Left*: all 256  $v2k7$  tcodes in 72 clusters. *Right*: a random sample of 2200  $v2k7$  rcodes, but biased for a representative distribution over  $\lambda_{ratio}$  and  $Z$ .

$Z_{left}$  and  $Z_{right}$  are components of the  $Z$  parameter.  $Z$  itself is greater of the two [31]. Whichever plot is selected, the prompts that follow are similar. Enter **return** for a  $Z - \lambda_{ratio}$  plot, or **1** for a  $Z_{left} - Z_{right}$  plot.

#### 24.9.4.1 $Z - \lambda_{ratio}$ or $Z_{left} - Z_{right}$

There are two methods of predefining the rule set,

countdown ... from the (start) rule selected in section 16.1.1. 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, 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  $\lambda_{ratio}$  and  $Z$  — for rcodes  $k \geq 4$ .

#### 24.9.4.2 $Z - \lambda_{ratio}$ or $Z_{left} - Z_{right}$ for $k \geq 4$ rcode

For rcode where  $k=4$  or  $k=5$ , the following prompt allows the rule sample size to be specified, as well as the countdown method,

**lambda-Z graph: countdown-C, number of rules (def 2200):**

or

**Zleft-Zright graph: countdown-C, number of rules (def 2200):**

Enter **C** for countdown as described in section 24.9.4.1 – this may take some time for  $k=5$  where rule-space =  $2^{32} = 4294967296$ . 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  $\lambda_{ratio}$  and  $Z$ . The following further option allows an unbiased random sample.

**random rule sample will have lambda bias, no bias-n:**

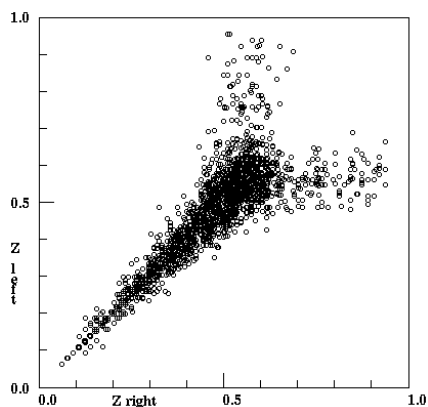


Figure 24.15:  $Z_{left}$  plotted against  $Z_{right}$  for 2200  $v2k7$  rcodes selected at random but with a bias to cover a representative spread of  $Z$ .

An unbiased sample is useful in estimating the proportion of rule-space with different measure of the  $\lambda_{ratio}$ ,  $P$ -parameter and canalizing inputs. Once the plot is complete, a lower window prompts for these details, described in section 24.9.5.

#### 24.9.4.3 $Z - \lambda_{ratio}$ 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 16.1.1. Because each tcode must be transformed to rcode, computation can be lengthy for large  $k$ . These options allow skipping (and seeing) equivalent rules.

Enter **a** to compute all rules (without skipping). Enter **e** to show one representative tcode from each set of equivalent tcodes. Enter **return** for the default — one tcode from each cluster<sup>4</sup>.

A further prompt allows a pause to see the equivalents of members of the tcode cluster,

**pause to show codes-p:**

If **p** is entered above, after each point is plotted, data about equivalent tcodes and clusters will be displayed in a top-right window, for example, for  $k=7$  tcodes and countdown from the maximum tcode 255,

**v2k7-tcode:233=104 22=151 equiv=2 clust=4 tot=22 cont-ret:**

This signifies that tcodes 233 and 104 are equivalent, as are their complements, 22 and 151. These 4 tcodes make up a cluster. In this example, the number of clusters plotted so far is 22.

Once the plot is complete, a top-right window gives information about the frequency of canalizing levels, for example for  $v2k3$  rcode (elementary rule-space),

**c-frequency:136 78 24 18 total rules=256 last=256**

This shows the frequency of rules with the 4 possible canalising levels (0, 1, 2, 3) — because  $k=3$ . A lower window prompts for more detailed parameter frequency information, described in section 24.9.5.

<sup>4</sup>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 complements of both equivalents [31]. Rules in a rule cluster have the same measures of  $Z$ ,  $\lambda_{ratio}$  (and also  $G$ -density) so the resulting plots will be identical.

## 24.9.5 Proportions of canalizing inputs and $\lambda_{ratio}$ - $P$

Once the  $Z - \lambda_{ratio}$  or  $Z_{left} - Z_{right}$  plot in section 24.9.4 is complete, data on the proportions of rules in the rule sample with different frequencies of canalizing levels ( $0 - k$ ), and parameters  $\lambda_{ratio}$  and  $P$  (which are equivalent), 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.

```
Canalizing-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,  $\lambda_{ratio}$  and  $P$  data for  $v2k3$  rcode (elementary rule-space) with 256 rules.

```
Canalizing-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,  $\lambda_{ratio}$  and  $P$  data for  $v2k44$  rcode rule-space, 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  $k+1$  ( $0 - k$ ) 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  $\lambda_{ratio}$  as a fraction.

ldr (second line) ... as above shown in decimal.  $\lambda_{ratio}$ = the fraction 0s or 1s (whichever is less) relative to half rule-table size,  $\lambda_{ratio}$  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  $\lambda_{ratio}$  as follows,  $P = 1 - (\lambda_{ratio}/2)$

- freq ... the number and percentage of rules that were found in each  $P$  or  $\lambda_{ratio}$  category above.
- C=0, C=1, C=2, ... the number and percentage of each canalyzing level that was found in each  $P$  or  $\lambda_{ratio}$  category above — numbers shown if less than 999.

## 24.10 “Backwards” space-time patterns

At the same time that attractor basins 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, which 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 (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 leaves (garden-of Eden states) — a leaf state has no pre-images thus no subtree.

The initial default is not to show the backward space-time patterns — the following top-right prompt is presented in section 24.4 to toggle the current setting (enter **s**).

**backwards space-time patterns (now OFF) tog-s: (or now ON)**

### 24.10.1 Scroll space-time patterns

By default, 1d 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 24.10, 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 24.12).

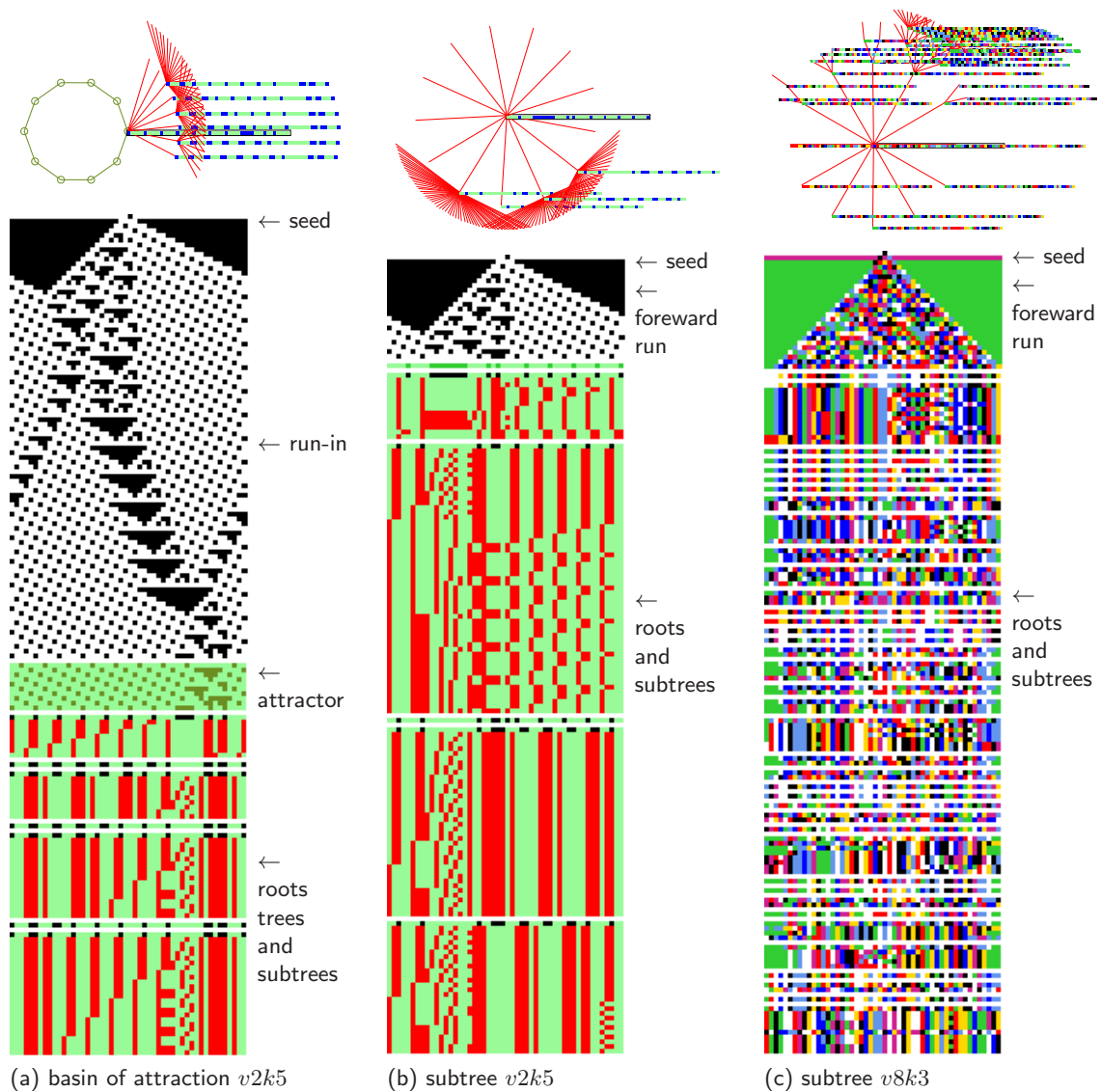


Figure 24.16: “Backwards” space-time patterns ( $n=50$ ) from a singleton seed, also showing (at top) the basin or subtree state transition graph with the bit/value patterns at nodes. (a,b)  $v2k5$  rcode (hex)e0897801. (a) shows a basin of attraction, (b) a subtree from a state 22 time-steps forward from the original seed. (c)  $v8k3$  modified chain rule (rcode), 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 roots, and their preimages. The figures show just the start of the backward run, which may continue for an indeterminate length of time.

## 24.11 Basin speed

Drawing basins and backwards space-time patterns may happen too fast for some purposes, such as the key hit on-the-fly options in section 24.12 where slowing down is done with the < key – so slowing down may need to be preset.

The “speed” option in section 24.4 has the following top-right prompt,

**speed: max-(def), or slowmotion (try 9000):**

Enter a number to slow down, the higher the slower. Maximum speed can be restored on-the-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 slow motion is to pause backward iteration after each tree or each pre-image fan (section 27.1.2).

## 24.12 Basin on-the-fly options

*refer also to section 30.3*

While attractor basins and backwards space-time patterns are being drawn, a number of settings can be changed on-the-fly. A reminder similar to this will appear in the bottom title bar (section 5.5),

**matrix-m STP:tog/scroll/exp/contr-s/#/e/c slow/max-</>**

[DDLab ©1993-2016 advance-ret back/interrupt-q matrix-m STP:tog/scroll/exp/contr-s/#/e/c slow/max-</>](#)

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. Section 30.3 describes the options.

# Chapter 25

## Layout of attractor basins

*not in TFO-mode.*

Basins of attraction (and sub-trees) are drawn either singly, or in groups — for the basin of attraction field<sup>1</sup>, 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, described in this chapter, may be altered (or defaults accepted): the basin scale, new variables controlling the gap between successive transient levels, positions, angles, horizontal and vertical spacing, rows staggered, spacing increase (for a range of network size  $n$ ), and the right edge border. 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 “basin parameter” prompt in section 24.1 (or if the basin parameters 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).

---

### 25.1 The layout preview

The layout preview shows the anticipated size and positions of attractor basins, including the diminishing projection of transient levels from the attractor, as they would appear on the screen according to the current settings, and is presented 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 (radius  $R_0$ ) is the maximum attractor diameter. Successive circles indicate successive levels in transients. The first, top left, basin representation shows the first  $L$  “active” levels (default=90) where the gap between successive levels decreases asymptotically parameterised by  $F$  (default=1), thereafter the gap between further levels remains constant. Attractor basins illustrated in this book follow these default settings, however, new options (section 25.2.2) allow these parameters to be varied, permitting a wider diversity of basin presentation.

---

<sup>1</sup>The basin of attraction field can also be redraw within the attractor jump-graph nodes, according to the jump-graph layout (section 20.7) providing many additional layout possibilities to those described in this chapter.

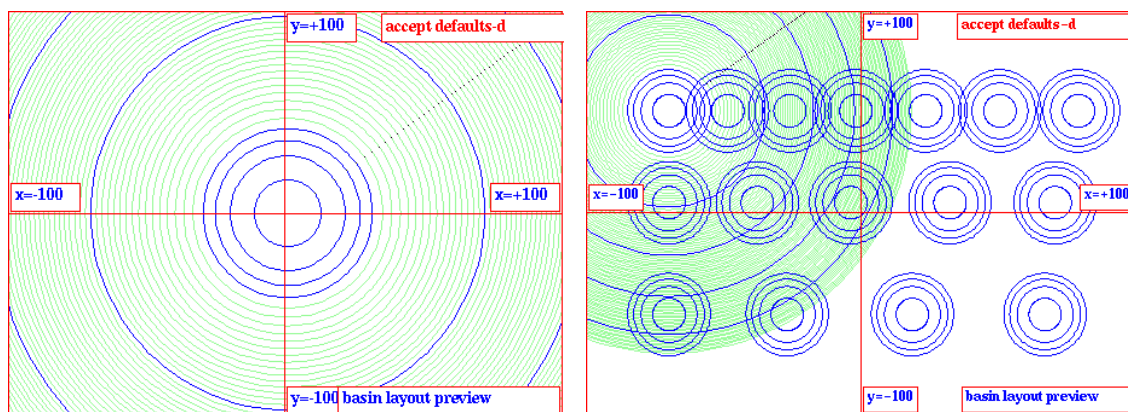


Figure 25.1: The default layout previews, *Left*: for a single basin or subtree, and *Right*: for a range of single basins.

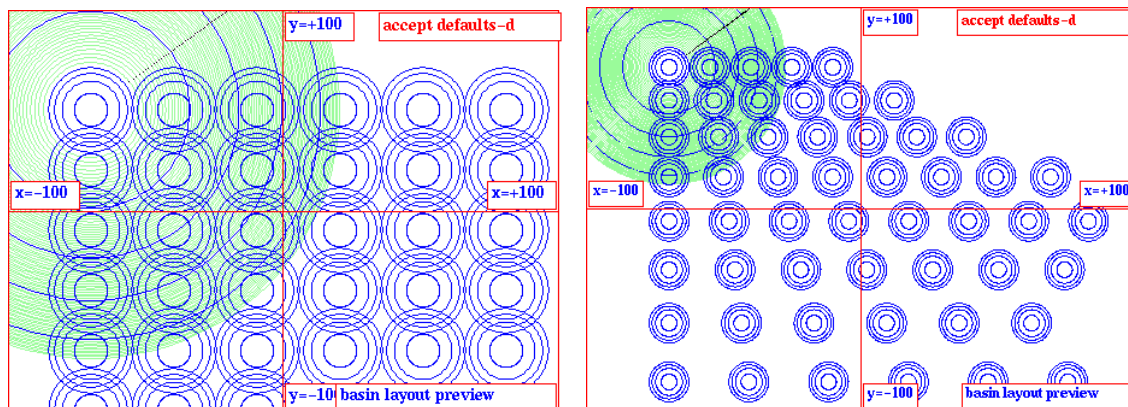


Figure 25.2: The default layout previews, *Left*: for a basin of attraction field, and *Right*: for a range of basin of attraction fields.

In the layout previews (figures 25.1 to 25.3) the first basin has every 20th circle highlighted, as well as the attractor circle itself and the first 3 levels. The 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.

## 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 (figures 25.1 to 25.3), together with the following (top-right) first layout prompt,



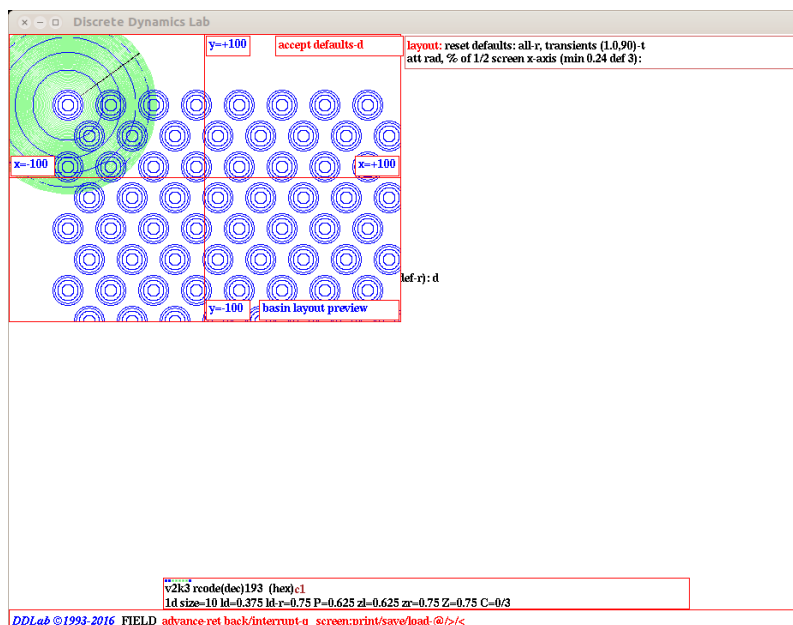


Figure 25.3: The layout preview as it appears on the DDLab screen. This example is for a basin of attraction field,  $\text{rad}=3$ , start position:  $x=-70$ ,  $y=50$ , spacing:  $x$ ,  $y$ , and right border= $22$ , with staggered rows.

**layout:** reset defaults: all-r, transients (1.0/90)-t, mutants-M (*M for single basins only*)  
 att rad, % of 1/2 screen x-axis (min 0.24 def 6):  
 (min depends on screen resolution, def on previous selections)

### 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 Reset transient scale

The transient scale can be reset, independently from the basin scale and attractor radius in section 25.2.4. Transient nodes are positioned on notional concentric circles shown in green and blue in the layout preview. The inner blue circle radius ( $R_0$ ) is the maximum attractor diameter. Successive circles away from the center mark transient levels — their projection from the attractor or subtree root. The first layout prompt in section 25.2, **transients-t (1.0,90)** gives the current transient parameters ( $F, L$ ).  $F$  (original default=1) controls the asymptotic decrease in the gap between successive transient levels<sup>2</sup>, and  $L$  (original default=90) gives the number of these “active” levels to which the decrease applies — further gaps then remain constant.

<sup>2</sup>The asymptotically decreasing gap between levels is calculated from the radius of successive levels  $(R_0 \times F) / \log(l^2)$  where  $R_0$  is the start radius,  $l$  is the current level, and  $F$  is a parameter (original default=1) which can be varied.  $F > 1$  will reduce, and  $F < 1$  will enlarge, the extent of transients. In either case the transient gap will decrease for  $L$  levels then remain constant equal to the last gap.

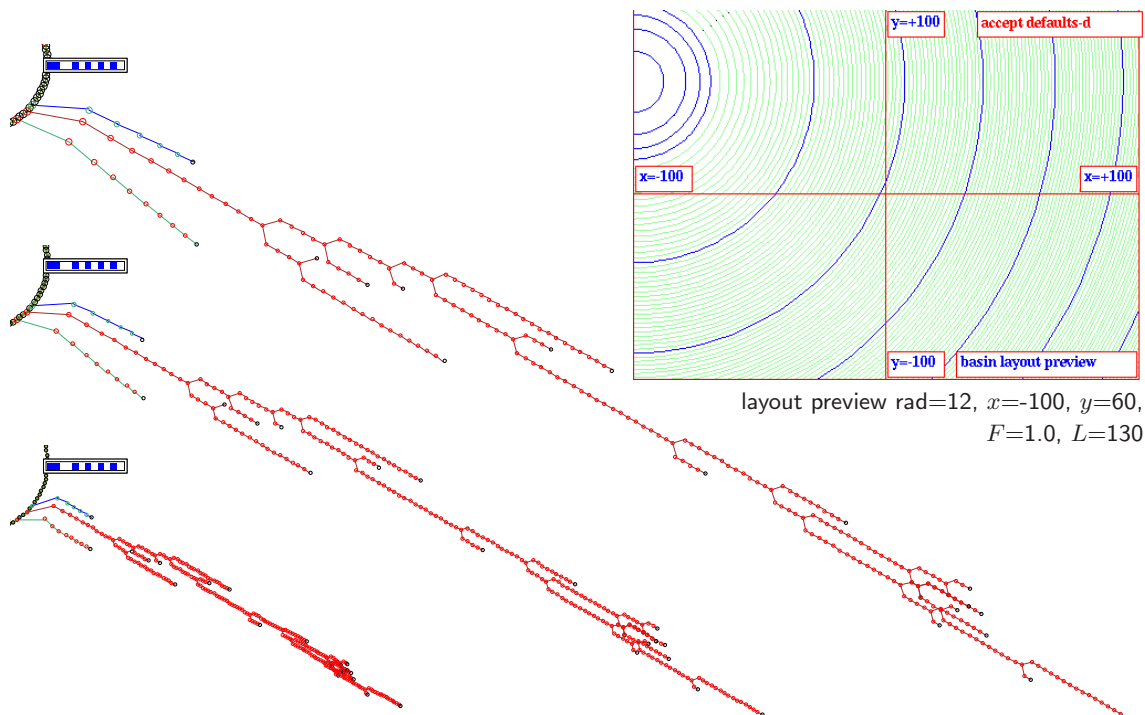


Figure 25.4: Resetting transient parameters for a single basin of attraction, `v2k3 rcode (dec)30, n=12, seed=(hex)0baa` — a chain rule with long transients showing just a segment of the attractor (period 102) with a state highlighted (see the whole basin in figure 2). *Top Right*: the layout preview. Active levels  $L=126$  were set to accommodate the longest transient. Three settings of the parameter  $F$  controlling the asymptotic gap decrease are shown. *Top*:  $F=1.0$  (the original default), *Center*:  $F=1.5$ , *Bottom*:  $F=3.0$ . The pre-image fan angle was also increased by setting its factor to 3.0 in section 26.4.2.

**layout:** reset defaults: all-r, transients (1.0,90)-t, mutants-M (M for single basins only)  
 att rad, % of 1/2 screen x-axis (min 0.24 def 6):  
 (min depends on screen resolution, def on previous selections)

Enter **t** to reset these parameters<sup>3</sup>. The following (top-right) prompt is presented,

**transient defaults: reset original (F=1.0, L=90)-r** (the original defaults)  
**amend: F (now 4.0): L (now 222):** (current values — for example)

Enter **r** to restore the original defaults, or enter the new values for  $F$  and  $L$  in succession, which remain as the defaults for the current basin session until DDLab is backtracked to the main prompt sequence — then the original defaults (1.0,90) are restored. Except for the examples noted, these original defaults are applied for the attractor basins illustrated throughout this book. However, for very long transients characteristic of chaos, as in “chain” rules, it may be useful to curtail the extent of transients by increasing  $F$ , and increasing  $L$  to include more active levels, for example figures 2 and 25.4 above. On the other hand, the opposite changes may improve the look of short transients — its really a matter of taste.

<sup>3</sup>Another significant transient default that can be altered is the pre-image fan angle (section 26.4.2).

### 25.2.3 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, figure 28.3).

### 25.2.4 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 sometimes required for basins with very long transients, though this can be adjusted in section 25.2.2.

### 25.2.5 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 center**  
**x (def -70):      y (def 50):**      (*values shown are examples*)

Enter the new polar coordinates for the first basin, first  $x$ , then  $y$ . As shown in figure 25.1, 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<sup>4</sup>, allowing, for example, the relevant part of a large basin to be displayed.

### 25.2.6 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 25.2.5) 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 27.2.3) – enter **return** will generate the next basin.

### 25.2.7 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:

<sup>4</sup>For groups of attractors in successive rows (not a single basin), their centers may end up below the visible screen. If within one screen height below they will be drawn as usual — though just the tip of a transient may be visible. Below this limit they are repositioned on the limit, and their transients are not drawn.

**basin spacing, % of 1/2 screen** (defaults depend on the radius set in section 25.2.4)  
**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

### 25.2.8 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,

**select right border as % of x spacing (def 45.0%):** (def depends on spacing in section 25.2.7)

### 25.2.9 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 increase 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,

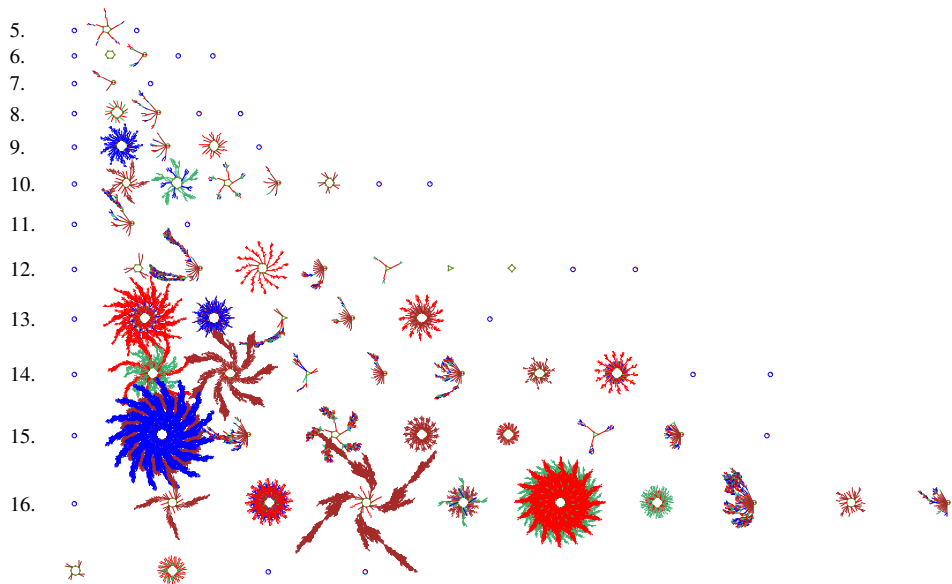


Figure 25.5: The basin of attraction field for a range of network size,  $n=5-16$ , shown on the left,  $v2k4$  rcode (hex)8b26. The spacing between basins increases at a set rate as larger basins are expected for greater  $n$ . In this example, the basin scale is 1.2 (section 25.2.4), the  $x, y$  start position is  $-90, 70$  (section 25.2.5), the initial  $x, y$  spacing between basins is 6,6 (section 25.2.7), the right border width is 45% (section 25.2.8), and the  $x, y$  spacing increase is set at 12%.

**% increase in spacing for successive network size,  
x increase (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 25.5 where the default setting were slightly amended, and in figure 4.5.

### 25.2.10 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,

**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 27.1.2, 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-center window (section 27.1.3) 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 27.1.3) — 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)*

These options can also be preset in section 26.4 where further details are given.

### 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 25.2.7 and 25.2.8 above. Figures 25.6 and 27.1 give examples.

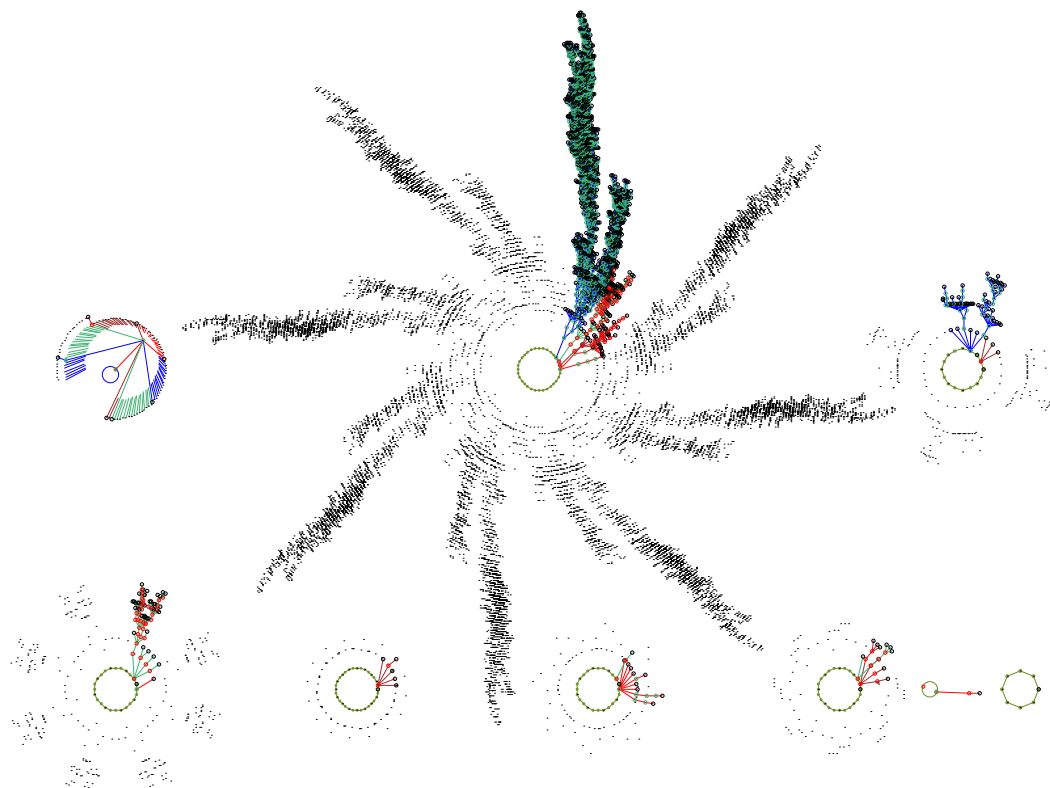


Figure 25.6: Amending the layout of a basin of attraction field on-the-fly, for a CA  $v2k3$ , rcode (dec)110,  $n=16$ . The field is shown with copies of equivalent trees suppressed except for garden-of-Eden nodes shown as dots (section 26.2.3). Layout parameters were set as follows: basin scale 3.8 (section 25.2.4),  $x, y$  start position -75,13 (section 25.2.5), initial  $x, y$  spacing between basins is 65,73 (section 25.2.7), right border width 15%, (section 25.2.8). A pause after each basin was set (section 27.1.2) allowing the  $x$  spacing to be changed — after the 4th basin to 40, after the 7th basin to 15.

### 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 25.2.5 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 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. See section 25.2.5 for further details about these settings. Figures 25.6 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 25.2.7 and 25.2.8 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 25.2.9.

---

# Chapter 26

## Display of attractor basins

*not in TFO-mode*

This chapter describes various options to change the default display of attractor basins, and to set null boundary conditions (NBC), as opposed to periodic boundary conditions (PBC).

Enter **p** at the first “output parameter” prompt in section 25.2 to jump directly to the display/NBC prompts, or reach them by viewing the output parameter in sequence.

For PBC only, and local 1d or 2d CA, attractor basins are “compressed” by default (section 26.2).

---

### 26.1 Attractor basins with null boundary conditions

*for 1d networks only*

As described in section 2.7, as well as the default Periodic Boundary Conditions (PBC), new options in DDLab allow Null Boundary Conditions (NBC), where inputs beyond the network’s edges are held at a constant value of zero. Although NBC for space-time patterns (running forward, section 31.3) apply for 1d, 2d and 3d networks, for attractor basins (running backward) NBC is restricted to 1d networks only.

Three completely different reverse algorithms are applicable, where the original PBC algorithms have been modified for NBC: (1) for CA, (2) for RBN/DDN as well as CA, and (3) the exhaustive reverse algorithm for any of the above, thus providing a reality check of results. Provided the network is 1d, all (backward) methods and functions for basin of attraction fields, single basins and subtrees apply.

The following top-right prompt gives a reminder of the network: local/non-local 1d,  $v$ ,  $k$  and  $n$ , and offers the new option for NBC,

**display:** local 1d CA, v2k3 size=10 (for example)

**Periodic boundaries (compression ON, off-s:) Null-N:**

Enter **N** to set NBC, the top-right prompt changes to the following,

... (as before, or non-local network if RBN or DDN)

**Null boundaries (compression OFF) Periodic-P:**

Enter **return** to accept NBC, or **P** to revert to PBC.



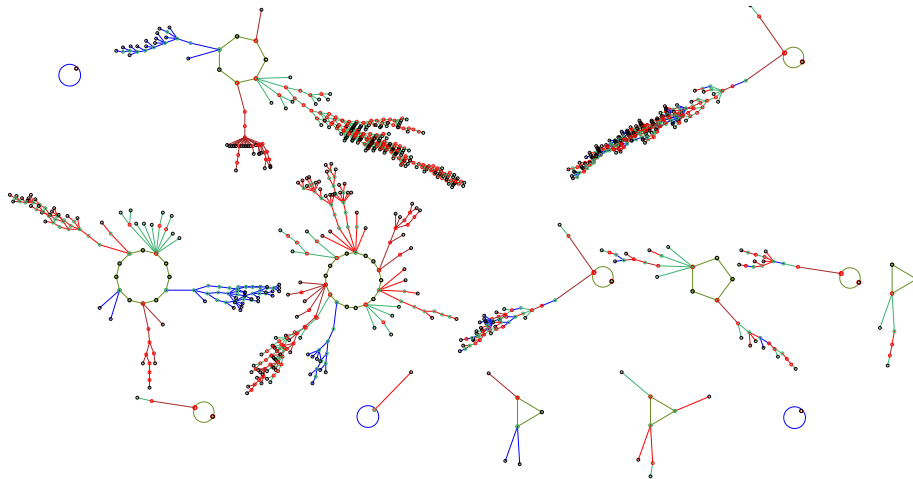


Figure 26.1: A basin of attraction field of a 1d CA with null boundary conditions (NBC)  $v2k3$   $r_{code}(dec)110$ ,  $n=10$ . Compression does not apply with NBC, and the basins are very different from the same network with periodic boundary conditions (and no compression) in figure 26.4. In this example the basins were positioned within suitably rearranged jump-graph nodes (section 20.7).

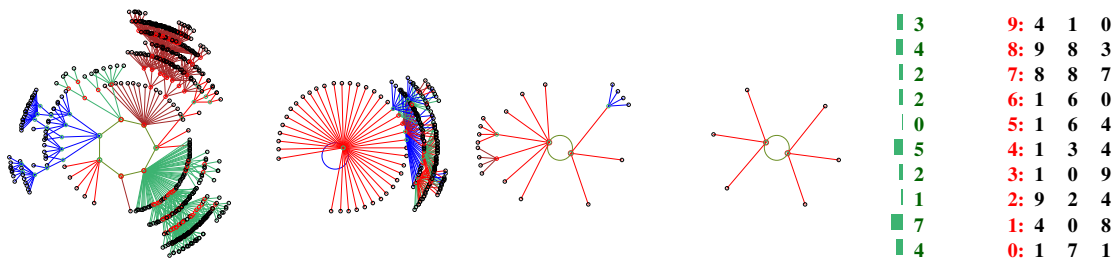


Figure 26.2: A basin of attraction field of a 1d RBN with null boundary conditions (NBC)  $v2k3$  homogeneous  $r_{code}(dec)110$ ,  $n=10$ . Right: The wiring, set out as in section 17.2.

## 26.2 Compression of equivalent CA dynamics for periodic boundaries

If the network is a local 1d or 2d CA (i.e. CA wiring and just one rule) with periodic boundary conditions, rotational symmetries of the periodic array result in equivalent dynamics [31]. Given a network state (periodic pattern),  $A$ , embedded in a particular past and future made up of other time-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

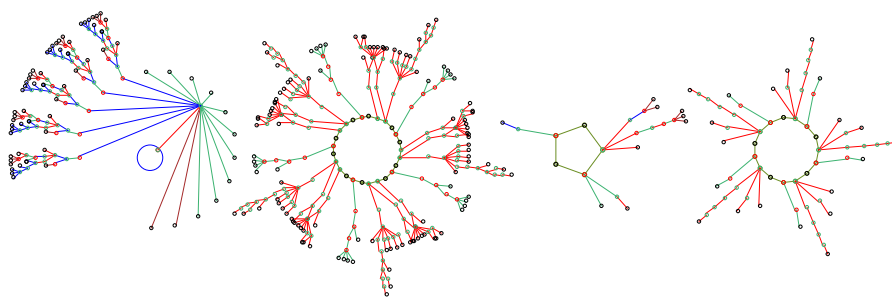


Figure 26.3: A basin of attraction field with compression of rotational equivalent dynamics,  $v2k3$  rcode(dec)110,  $n=10$ . Only one prototype of each set of equivalent basins is drawn. In addition, equivalent trees on attractor cycles (2nd and 4th basin), and subtrees from a uniform states (1st basin) are copied and rotated, thus further speeding up computation as a reverse algorithm need not be re-applied.

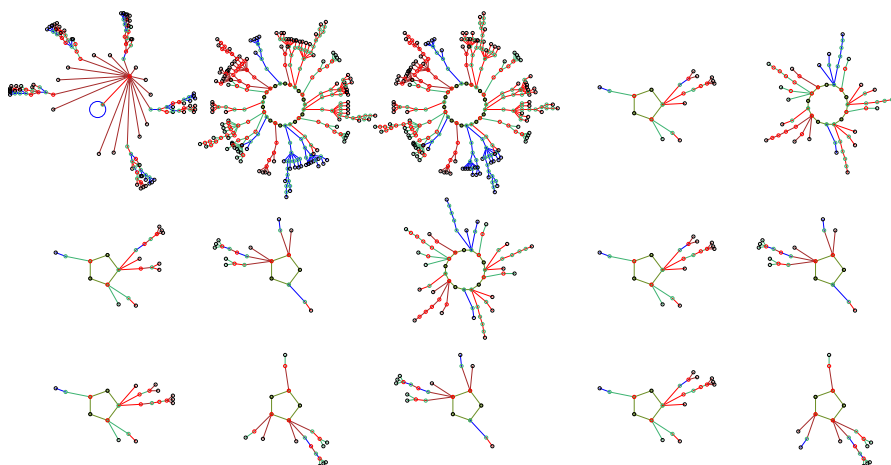


Figure 26.4: The same basin of attraction field as in figure 26.3 without compression of equivalent dynamics. Compression was disabled so that all components of the state transition graph are computed from scratch with a reverse algorithm, not copied and rotated as in figure 26.3. All states in state space are represented.

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.

It was proved in “The Global Dynamics of Cellular Automata” [31] 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* where all values are equal (all 0s and all 1s for binary) with the shortest repeating segment size ( $s=1$ ) are often powerful attractors.

For local 1d and 2d CA, “compression” algorithms automatically come into play (unless disabled in section 26.2.1) 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.

### 26.2.1 Deactivate compression

For 1d or 2d CA with periodic boundary conditions only, the default has compression active (section 26.2), but this can be deactivated. The top-right prompt gives a reminder of the  $v$ ,  $k$  and  $n$  (or  $i \times j$  for 2d), and is presented as follows, with a new option for null boundary conditions,

**display:** local 1d CA, v2k3 size=10 (for example)  
 Periodic boundaries (compression ON, off-s:) Null-N:

Enter s to deactivate the compression algorithms.

### 26.2.2 Pre-images of uniform states

The uniform states, where all cell values are equal, enjoy a privileged status in the dynamics of CA — they have maximum rotational symmetry (RS) so lie on or very close to attractors.

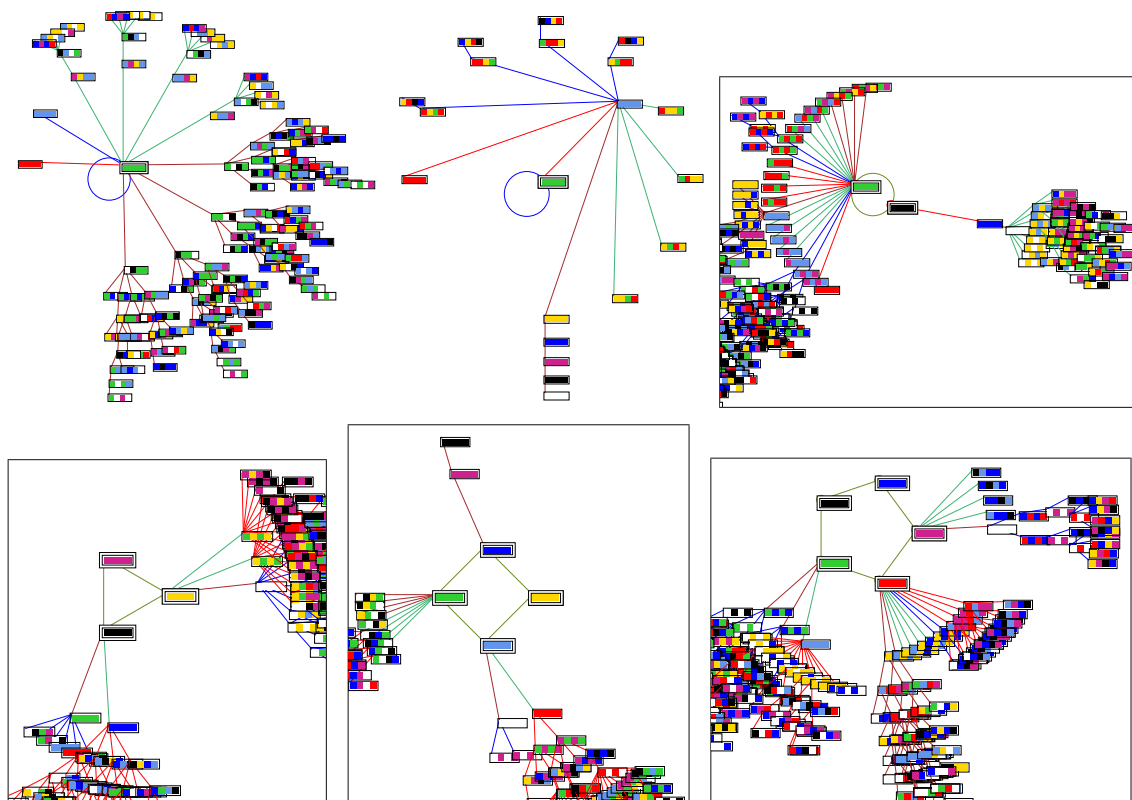


Figure 26.5: The special properties of CA uniform states, which lie on or very close to attractors, allow speeding up computation of their pre-image fan and attached trees. Each (sub)tree type only needs to be computed once, the equivalent (sub)trees simply have their states rotated. The pre-image fan is organized into equivalent groups. This is illustrated here by some 1d CA  $v8k2$  rcodes  $n=4$ , 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. The attractor periods shown range from 1 to 5. The framed basins are fragments centered on the attractor.

Uniform states can only be pre-images of each other<sup>1</sup>. States with higher RS must be downstream of states with lower RS — all states in an attractor must have equal RS [31]. 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 26.2) as well as organizing 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 26.5, 26.10). In principle this could be extended to any uniform state, and also to states with repeating segments in general — this is not implemented in DDLab at present.

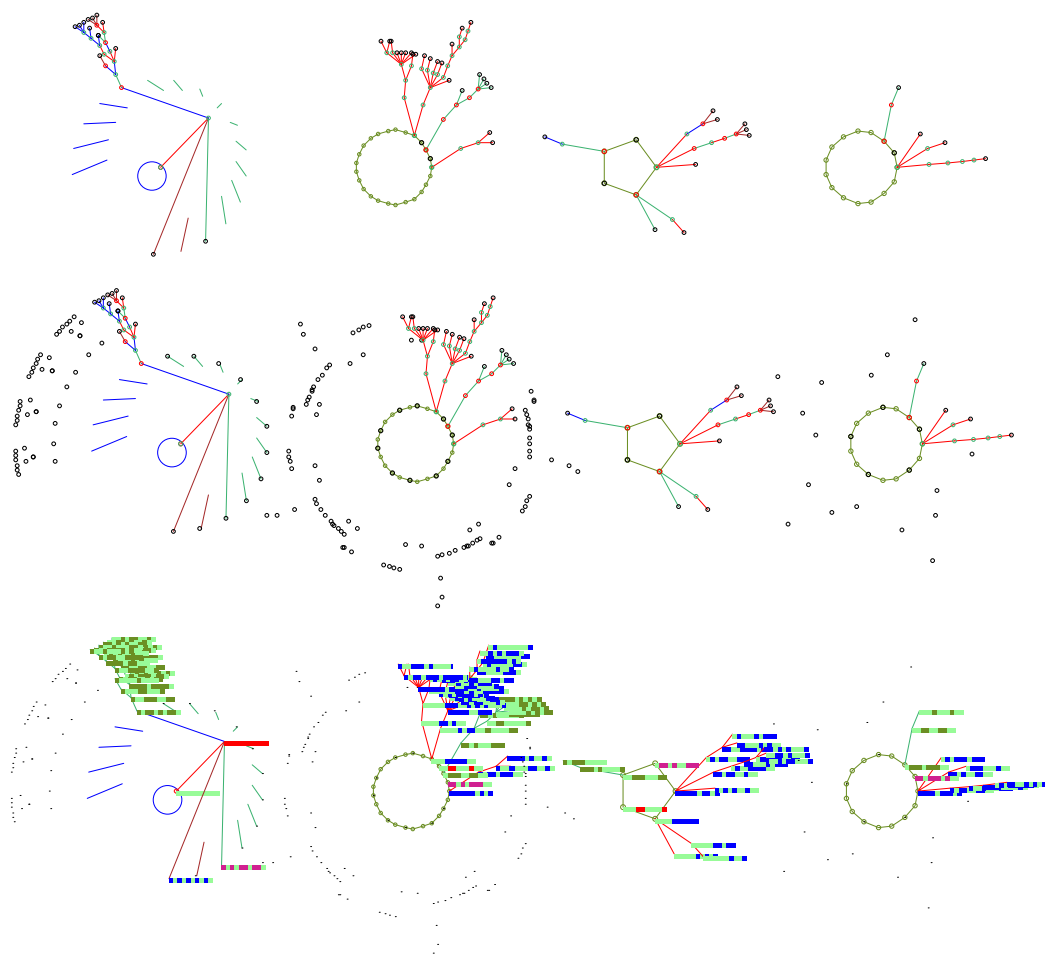


Figure 26.6: Suppressing equivalent subtrees in a basin of attraction field,  $v2k3$  rcode(dec)110,  $n=10$ , (as in figure 26.3). *Top*: all equivalent subtrees suppressed, however, short edges are still shown from a uniform state (1st basin). *Center*: leaf states in the suppressed subtrees shown as spots, and *Bottom*: as one pixel dots. Nodes in the prototype trees are shown as selected in section 26.3 — in the latter case as a 1d bit pattern.

<sup>1</sup>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.

### 26.2.3 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 transients (figures 25.6, 26.6). The prototype tree/subtree is drawn in full with whatever node display set in section 26.3. 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. Alternatively, 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.

## 26.3 Node display

The nodes in attractor basins may be displayed in a variety of ways — 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 26.8), 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 (options summarized 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):**

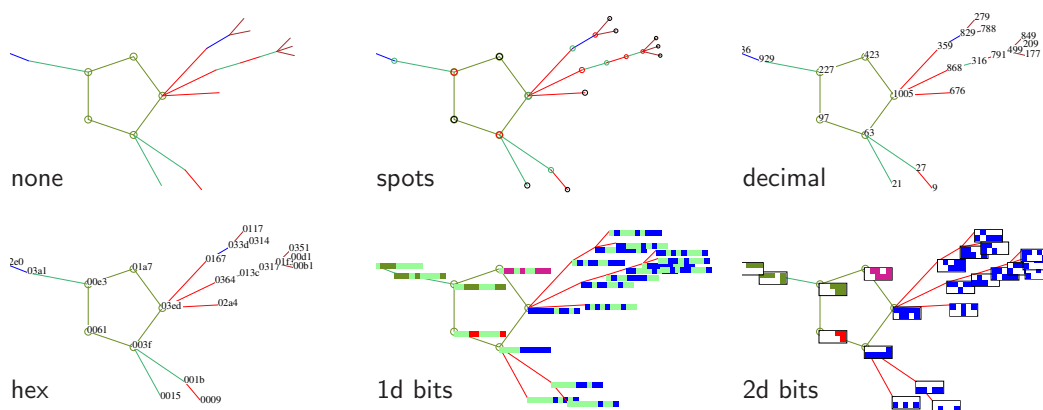


Figure 26.7: Examples of alternative node display. *from the Top-Left:* none (except on the attractor): spots, decimal, hex, 1d bit/value pattern, and 2d bit/value pattern, where the  $i \times j$  dimensions can be preset. This is the small basin in figures 26.3—26.6,  $v2k3$  rcode(dec)110,  $n=10$ .

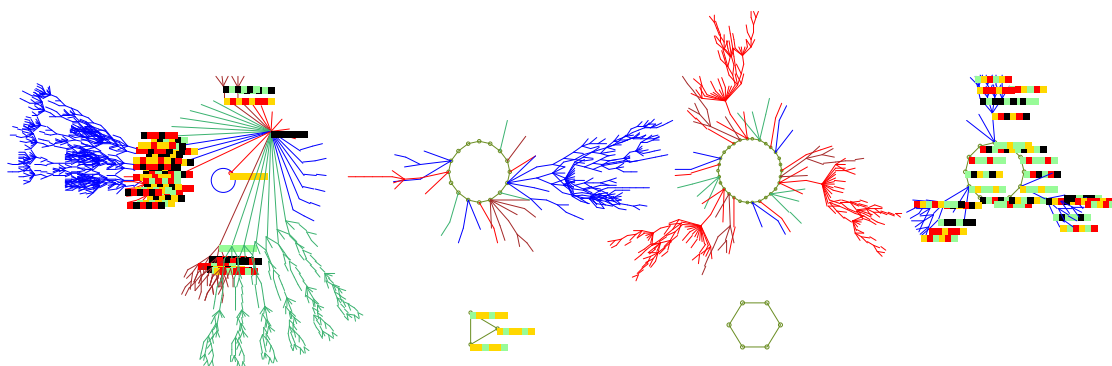


Figure 26.8: Symmetric states only, those made up of repeating segments (including the uniform states) may be shown while other nodes are suppressed (enter **bS**). Here, the symmetric states appear as value-pattern nodes in a basin of attraction field — they occur in basins 1, 4 and 5. *v4k3* CA,  $n=6$ , `rcode(hex) 4d3f8c86143ffca77a0ed4b5d172e7c3`.

For networks other than local 1d or 2d CA, this is the first prompt in the display sequence, preceded by the title **display:** — the prompts in sections 26.2.1 and 26.2.3 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 (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.

*+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 26.8.

**just g-nodes +g** ... add **g** for just leaf states (and attractor states without transients). For example enter **Bg** for figure 26.9Left.

**no g-nodes +G** ... add **G** to exclude leaf states (and attractor states without transients). For example enter **BG** for figure 26.9Right.

**none-n** ... to omit nodes altogether.

The options for node display can be combined with any other node options — compression (sections 26.2.1 – 26.2.3) and highlighting attractor and subtree root nodes (section 26.3.2).

The colors of spots cycle through four colors to contrast with transient edge colors (section 23.4.1). 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 26.3.1).

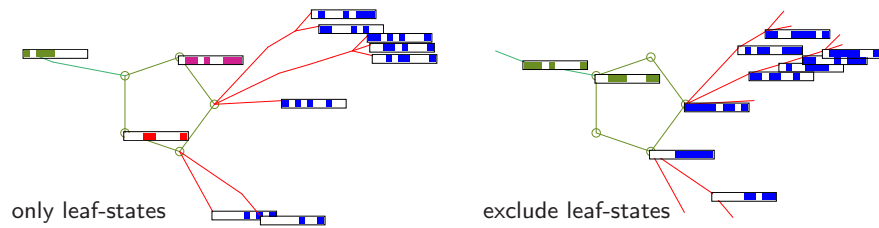


Figure 26.9: Showing just leaf-states or excluding leaf-states,  $v2k3$  rcode(dec)110,  $n=10$ , (as in figure 26.6). *Left*: only leaf-states and attractor states without transients are shown. *Right*: leaf-states and attractor states without transients, are excluded. The nodes are shown as 2d bit patterns  $10 \times 1$ .

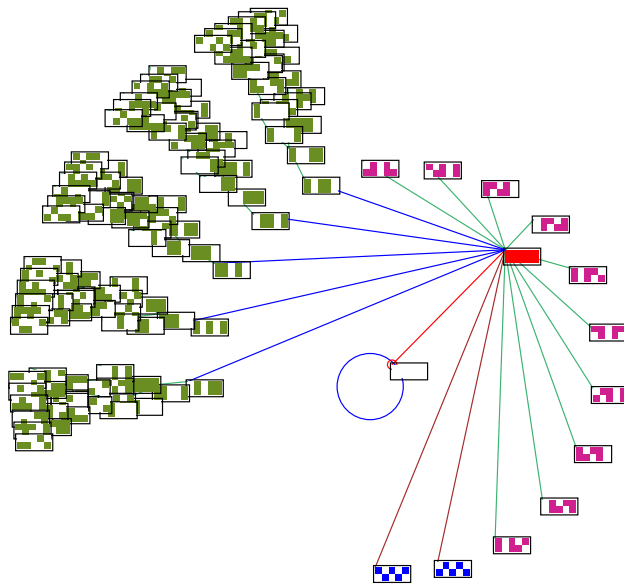


Figure 26.10: The pre-images of a uniform state's tree are organized into groups of equivalents. For binary systems, the bit patterns of these states, and states in their subtrees are shown in the same color, cycling through four colors. Nodes are shown as 2d bit patterns  $5 \times 2$ . This is the first basin in figure 26.3,  $v2k3$  rcode(dec)110,  $n=10$ .

### 26.3.1 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 23.4.1) 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 26.10). 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 26.2) 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.

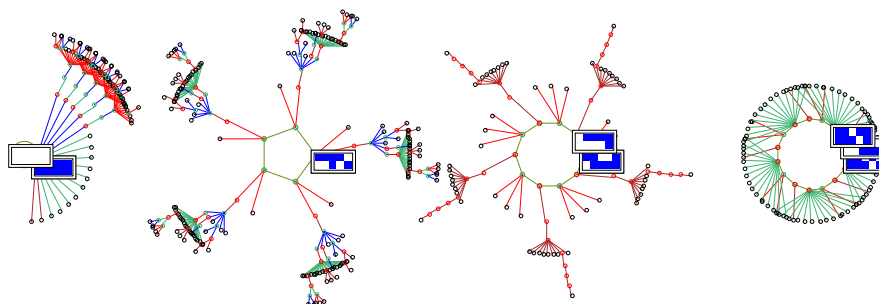


Figure 26.11: Highlighting all non-equivalent attractor states as a 2d bit pattern for the basin of attraction field of a 1d CA,  $v2k3$  rcode(dec)111,  $n=10$ . Other node are shown as spots.

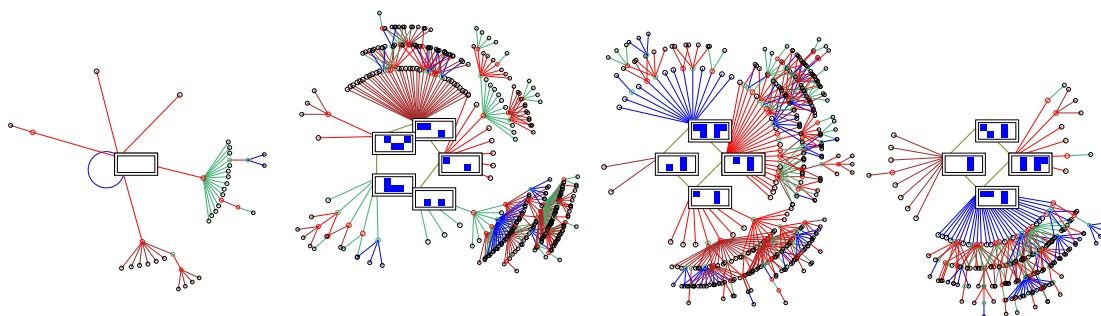


Figure 26.12: Highlighting all attractor states in a RBN basin of attraction field,  $v2k4$ ,  $n=10$ . Other node are shown as spots.

### 26.3.2 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 26.3. The following top-right prompt is presented,

**highlight attractor (or subtree root) in 2d: one-1 all-a:**

Enter **1** to highlight one attractor state — this state is the last attractor state reached in the initial forward run, and is positioned due east on the attractor cycle (figure 26.15).

Enter **a** to highlight all non-equivalent attractor states for local 1d or 2d CA (figure 26.11) – if compression is set in section 26.2.1. Otherwise if **a** is entered all attractor states will be highlighted, as in figure 26.12 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 26.14



### 26.3.3 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 26.3).
- 2d-b** ... for 2d bit/value patterns for 2d networks (section 26.3).
- 1** or **a** ... to highlight all attractor states, just one, or a subtree root (section 26.3.2).

A 2d network retains its  $i, j$  (a 3d network its  $i$ ) as the default, set in sections 11.6.2 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 21.4.7). For any dimension, 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 26.3.4.

### 26.3.4 Alter scale, divisions and dots — node as bits/values

If nodes are set as bits/values either in 1d or 2d (section 26.3) or if the attractor or subtree root nodes are highlighted in 2d (section 26.3.2), divisions between cells, dots at zero values, or both, can be added to the basic presentation, illustrated in table 26.1 below,


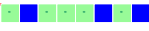


	basic	divs+dots
1d		
2d 5x2		

Table 26.1: Alternative presentations for bit/value nodes,  $v=2$ ,  $n=10$

For a cell scale in pixels  $p \leq 5$  the presentation without divisions or dots is the default. The scale can be changed, and divisions/dots toggled, with the following top-right prompts, which are presented in turn, 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, and will also be applied in a PostScript image (section 24.2). 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 24.10) and the cell scale of basin nodes as bit/value patterns described in this section, will be the same. Both can be changed on-the-fly as basin are being drawn (section 30.3).

### 26.3.5 Alter decimal or hex node scale

If nodes were set as decimal or hexadecimal in section 26.3 the default node label size is set automatically depending on the basin scale (section 25.2.4). 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:**

## 26.4 Change orientation, fan angle, edge color

The next three prompts allow the default orientation of attractor basins (section 23.3), the default pre-image fan angle, and the default edge colors, to be changed. The following top-right prompts are presented in turn,

**change basin orientation anti-clockwise (now 0), enter angle:**  
**change pre-image fan angle (now 1.00), enter factor:**  
**change start edge color (now 0) ----0 ----1 ----2 ----3:**

### 26.4.1 Orientation

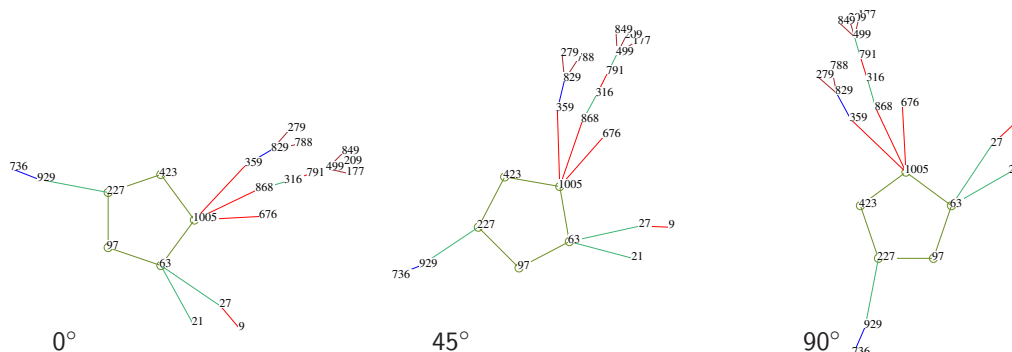


Figure 26.13: Changing the orientation of a single basin of attraction, starting with  $0^\circ$ , then setting  $45^\circ$ , and finally  $90^\circ$ . Labels are in decimal with a seed of 63,  $v2k3$  rcode(dec)110,  $n=10$ .

Changing the orientation allows attractor basins to be rotated anti-clockwise by the angle entered at the prompt in section 26.4,

**change basin orientation anti-clockwise (now 0), enter angle:** *(current angle shown)*

The default is  $0^\circ$  due east. Figure 26.13 gives examples. Note that for a single basin of attraction, with the orientation at  $0^\circ$ , 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).

### 26.4.2 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 26.14), or to increase the angle for chaotic rules with low branching (figure 26.15). This is done by entering a multiplication factor at the prompt in section 26.4,

**change pre-image fan angle (now 1.00), enter factor:** *(current factor shown)*

The fan-angle can also be changed during a pause as fields are drawn (section 30.2).

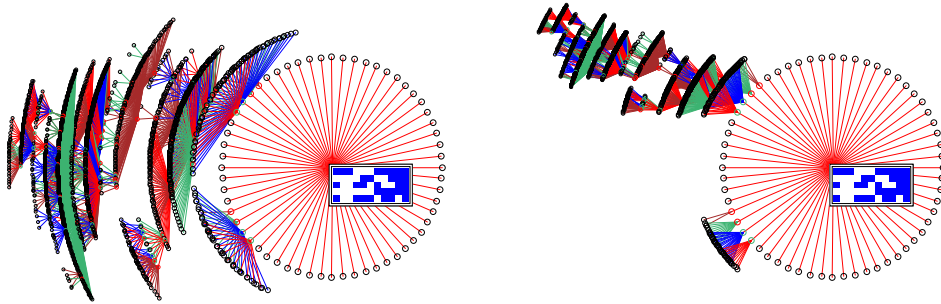


Figure 26.14: Decreasing the pre-image fan-angle of a subtree of a 1d CA  $v2k3$  `rcode(dec)110`,  $n=55$ . The subtree root, (hex)71f0cf34c0fced, is highlighted as a bit pattern. Other nodes are show as spots. *Left*: default fan-angle (factor=1). *Right*: reduced fan-angle (factor=0.3). Note that the fan converging on the root of a subtree is always  $360^\circ$ , and is not affected.

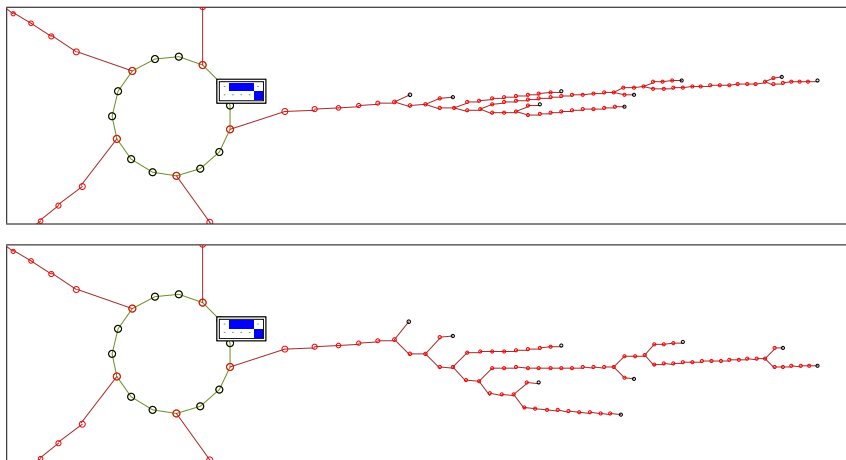


Figure 26.15: Increasing the pre-image fan-angle of a single basin of attraction of a 1d CA,  $v2k3$  `rcode(dec)30`  $n=10$ . This is a chain-rule with very low in-degree, characteristic of chaos. In each case, part of the basin including the attractor and one complete tree is shown. One attractor state 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  $34^\circ$  (section 26.4.1).

### 26.4.3 Edge color

As described in section 23.4.1, a cycle of four colors is used to draw transition edges, and this also determines node colors (section 26.3.1). The resultant attractor basin color scheme<sup>2</sup> depends on the start edge color (red is the default) which can be changed with this prompt in section 26.4, which also indicates the current color cycle,

**change start edge color (now 0) ----0 ----1 ----2 ----3:**

<sup>2</sup>Colors are different on a black background (section 6.3.3).

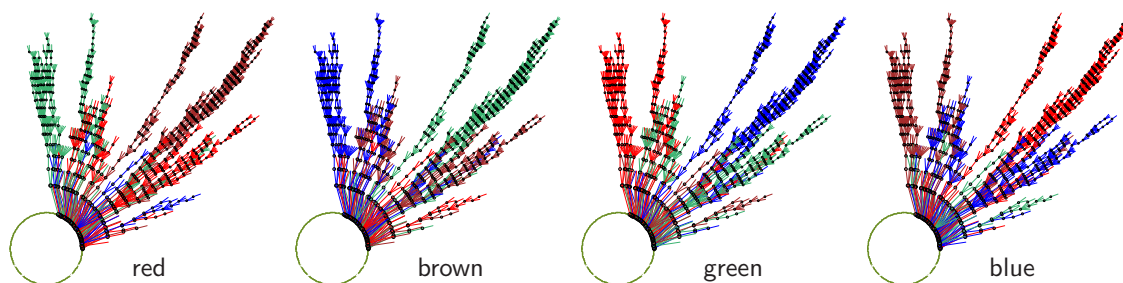


Figure 26.16: Alternative edge start colors (indicated) result in different basin color schemes. This is a single basin with copies of trees suppressed (section 26.2.3), the fan angle reduced by 0.5 (section 26.4.2) and leaf nodes excluded (section 26.3). Transient nodes are shown as spots. `v2k3 rcode(dec)110, n=15, seed(hex)660d`.

## 26.5 Limiting backward steps

It's 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. This option is part of “Final options for attractor basins” in chapter 29 and described in section 29.5.

For single basins the number of backward steps can be set at zero, showing just the bare attractor (section 29.5.1). Because the data is derived from forward iteration – reverse algorithms do not come into play — the graphic image of attractors for very large networks, including 2d, can be generated, as in figures 29.2 and 29.3.

# Chapter 27

## Pausing attractor basins, and data

*not in TFO-mode.*

This chapter describes methods for pausing attractor basins according to a hierarchy of stages during drawing (section 27.1), to show/save/print data at each stage (section 27.1.1), and activate other options including amending the layout (section 27.1.3). The level of data detail can be set in section 27.3, including a complete sorted list of all the states (section 27.5). Printing/saving the data can be done with or without pausing. The data is generated as attractor basins are drawn. The methods apply to any attractor basin type — a basin of attraction field, a single basin or a subtree, or a range  $n$  of the above.

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 attractor basins are complete, a data summary is displayed in any case, but setting a pause allows intermediate data during the drawing itself.

---

### 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 pause after each tree, a data pause will also occur at stages 2 and 1. Each stage gives a specific top-center pause prompt described in section 27.1.3.

### 27.1.1 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 27.1.3. If **f** was entered, the prompt in section 27.1.2 is presented for further stages of detail.

### 27.1.2 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, (enter **1**, **2** or **3** as required, or **return** not to pause),

*(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 single basin, or a subtree)*

**pause/data: pause for data: fan-3 tree-2, none-def:**

### 27.1.3 The pause prompt

At at each pause, as well as basin data in a top-right window, a prompt is presented in a top-center 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:**

*options ... what they mean*

- 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.

## 27.2 Attractor basin complete — data window

When an attractor basin is complete (or abandoned), data are displayed in a top-right window. Examples and decoding are shown below for a CA with compression active (section 26.2). The first item in this data window gives boundary conditions, either **PBC** (periodic), or **NBC** (null). The window also includes a top-right inset showing the type of wiring, algorithm or function that was utilised, as follows,

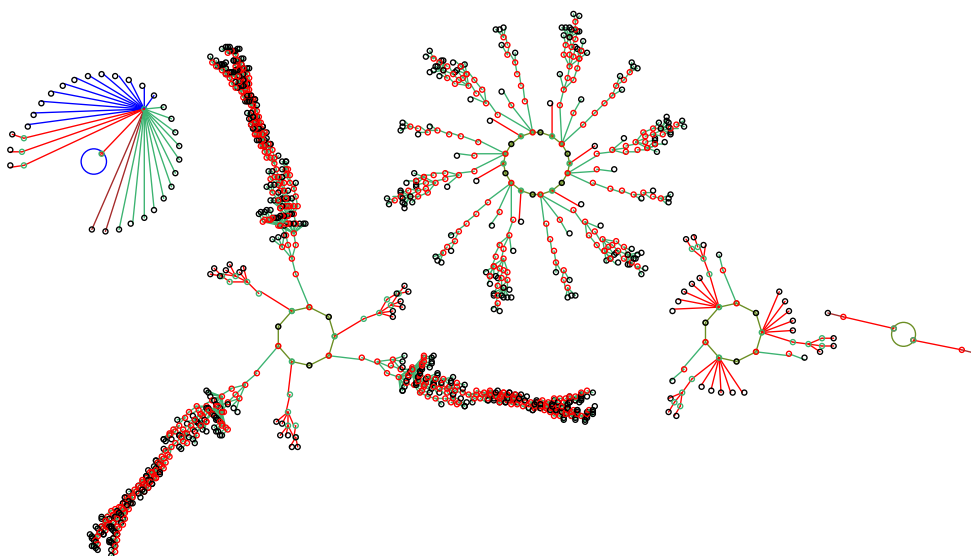


Figure 27.1: A basin of attraction field of a CA,  $v2k3$  rcode (dec)110,  $n=12$ , relating to various data outputs in section 27.2, and saved in sections 27.4.1 — 27.4.4. In this example the default layout was amended in the jump-graph with no edges (section 20.3).

*top-right insets ... what they mean*

- CAW** ... cellular automata wiring or local wiring.
- NLW** ... non-local wiring.
- EXH** ... the exhaustive algorithm.
- MAP** ... a random map, using the exhaustive algorithm.
- NTO** ... neutral order components.

More detailed data, including a sorted list of states, may also be printed or saved to a file (sections 27.3, 27.5).

### 27.2.1 Data on basin of attraction fields

By default, when a basin of attraction field is complete, unless a range of fields was specified (sections 8.1, 27.1.1), data on the field is presented in a top-right window. An example of data on a field (for  $v2k3$  rcode (dec)110,  $n=12$ ) as in figure 27.1 is shown below,

*for a single field, or if pausing in a range of fields*  
**PBC basin types=5 total basins=11**  
**n=12 field=4096 g=1971=0.481 0.109sec**

If the basin of attraction field is interrupted<sup>1</sup> (by entering **q** twice, see section 30.2), the data on the incomplete field appears as in the example below, depending on the point of interruption,

```
EARLY EXIT PBC 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.

**PBC** or **NBC** ... periodic or null boundary conditions.

**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 would indicate an error (section 27.2.2).

**g=x=y** ... the number and density of garden-of-Eden states in the field.

**xmin ysec** ... the time taken to draw the field.

## 27.2.2 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 (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.

## 27.2.3 Data on basins

An example of data on a basin (for *v2k3* rcode (dec)110,  $n=12$ ) is shown below<sup>2</sup>.

*for a single basin*

```
PBC 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*

```
PBC basin 2, equiv basins=4 att(hex)=00 73  
period=9 size=831 81.2% g=402=0.484 ml=35 mp=13
```

<sup>1</sup>In this and other examples, the computation was slowed (section 24.11) to give enough time to interrupt. Interrupts at other points in a basin or subtree are described in section 30.2.

<sup>2</sup>If a count of states with a majority of a given value is active, this data will also be displayed in the basin (or subtree) data window, for example **maj1=572=0.126** (section 24.7).



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 PBC 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.

**PBC** or **NBC** ... periodic or null boundary conditions.

**basin** *x* ... the basin in question is the prototype of the *x*th 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.

#### 27.2.4 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 length of the tree, excluding the attractor root.

**mp=** ... maximum in-degree found in the tree.

#### 27.2.5 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.

## 27.2.6 Data on subtrees

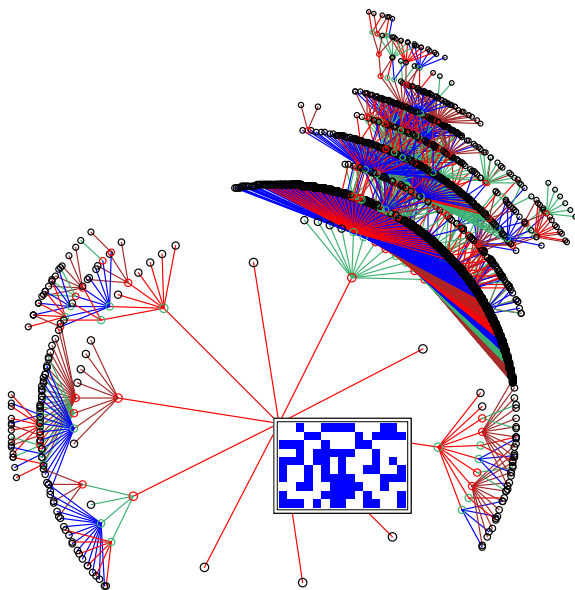


Figure 27.2: A subtree for a 1d CA, from a root state shown as a 2d  $15 \times 10$  bit pattern. The root was reached by iterated 3 steps forward from the seed state, which was chosen at random.  $n=150$ ,  $v2k5$  rcode(hex)aa5566a1.

If generating just one subtree 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 usually 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 — they are only shown if  $n \leq 56$ . This limit does not apply when data is printed or saved (section 27.4.7).

```
subtree=4335
g=4030=0.93 ml=11 mp=1253 5.292sec
```

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
```

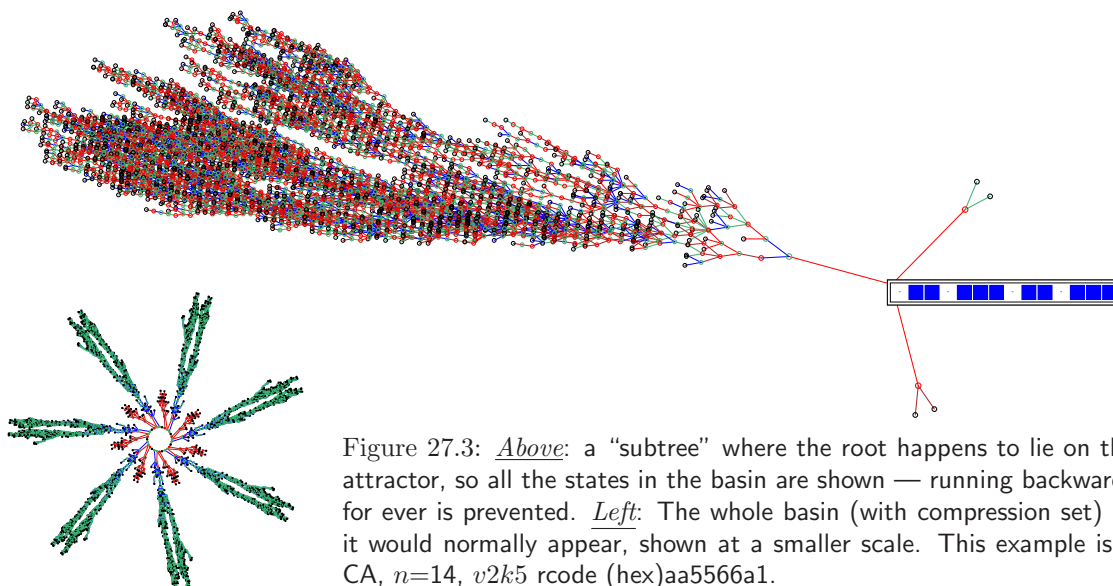


Figure 27.3: *Above*: a “subtree” where the root happens to lie on the attractor, so all the states in the basin are shown — running backwards for ever is prevented. *Left*: The whole basin (with compression set) as it would normally appear, shown at a smaller scale. This example is a CA,  $n=14$ ,  $v2k5$  rcode (hex)aa5566a1.

The second example (figure 27.3) is for a small CA  $n=14$  with the same rcode, 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 around the attractor and its trees.

```
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 subtree, 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.

**xmin ysec** ... the time taken to generate the subtree.

### 27.2.7 Data on subtrees from a uniform state

If a subtree pause is selected for a CA in section 27.1.2 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, and data will be displayed on each subtree prototype. An example of the data on the 4th prototype subtree in figure 27.4 is shown below,

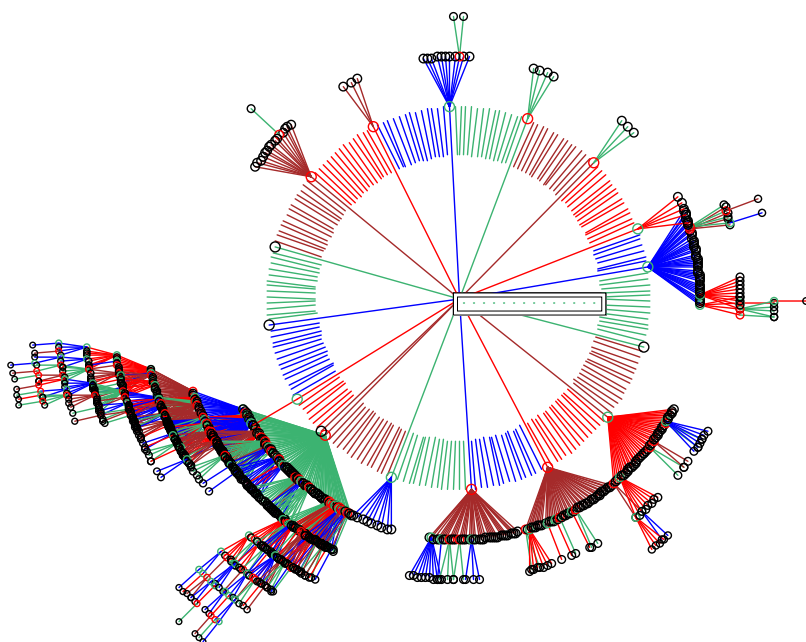


Figure 27.4: Prototype subtrees from a uniform state, all 0s, for a CA,  $v2k3$  rcode (dec)96,  $n=14$ . Copies of the prototypes have been suppressed in section 26.2.3, and the default fan angle reduced by a factor of 0.3 in section 26.4.2.

**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 subtree, 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 27.2.6),

**subtree=basin=16333 root(hex)=00 00**  
**g=13363=0.818 ml=9 mp=212 9.266sec**

## 27.3 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 27.2.1 — 27.2.7), and also a list of states in different formats (section 27.5) 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 27.5). 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,

*options ... what they mean*

**basin data only-1** ... for basin data (section 27.2.3) or subtree data (section 27.4.7).

**and tree data-2** ... for basin and tree data (section 27.2.6).

**include states:** ... *show just the states, or add states to selections above (section 27.5).*

**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 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 (section 35.3) – 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 27.4, together with the decoding of the data.

## 27.4 Data format

Examples of the data format, printed in the terminal (Linux-like systems only) or saved to a `.dat` ascii file (section 27.3) are described and decoded below.

### 27.4.1 Network parameters data

If **1** or **2** is selected in section 27.3, 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
z1=0.75 zr=0.625 Z=0.75 C=0/3
```

<i>data type</i>	<i>... what it means</i>
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	... $\lambda$ , $\lambda_{ratio}$ , $P$ -parameter.
z1=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.

### 27.4.2 Basin field data

An example of the complete data for the basin of attraction field in figure 27.1 (enter **1** in section 27.3),

```
data                                     ... decode
v2k3 rcode(dec)110 (hex)6e
1d size=12 ld=0.625 ld-r=0.75 P=0.625
z1=0.75 zr=0.625 Z=0.75 C=0/3    ... network parameters, described in section 27.4.1
above

ty.  at no (p) s % g gd ml mp    ... reminder of data order, key in section 27.4.3 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
```

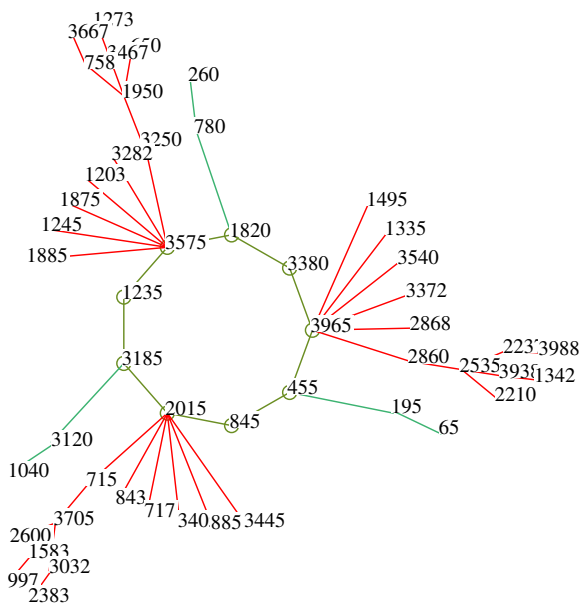


Figure 27.5: The 4th basin in figure 27.1 with nodes shown in decimal. CA  $v2k3$   $\text{rcode}(\text{dec})110$ ,  $n=12$ . To reproduce this basin and the data in section 27.4.6 generate this single basin with the seed  $(\text{dec})455$  or  $(\text{hex})01c7$ , and select  $\text{dec-c}$  in section 26.3

### 27.4.3 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 (section 26.2), or for a non-CA network where there are no equivalent basins — data on every basin will be shown.

### 27.4.4 Tree data

An example of the same data as for the basin of attraction field in section 27.4.2 above (figure 27.1), but also including tree (and subtree) data, which precedes the basin data summary is given below (enter 2 in section 27.3). For CA, subtree data is given for each subtree type rooted on the pre-images of a uniform attractor state (all values equal) in section 27.2.7.

```

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 27.4.1

ty.  at no (p) s % g gd ml mp    ... reminder of data order, key in section 27.4.3
     tree. root no s g gd ml mp ... reminder of tree data order, key in section 27.4.5
below
     seg-fan=29 (4 segs)    ... uniform state in-degree, and number of subtree types
     1.  Oaaa 2 1 1 1 2 0    ... data for each subtree type, in basin 1
     2.  Oab6 12 1 1 1 2 0
     3.  Oad6 12 1 1 1 2 0
     4.  Odb6 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 (figure 27.4.5)
     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 (figure 27.4.2)
     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

```

### 27.4.5 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 26.2), data is also given on subtrees from a uniform state. If CA compression is suppressed, or for a non-CA network where there are no equivalent trees — data on every tree will be shown.





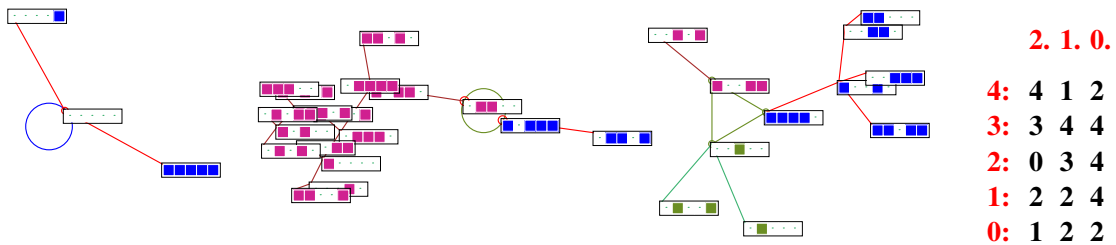


Figure 27.7: The basin of attraction field of an RBN, with states listed in section 27.5. The RBN has a homogeneous rule,  $v2k3$  rcode (dec)110,  $n=5$ . Nodes are displayed as a 2d bit pattern (section 26.3). *Right:* The wiring matrix (section 17.2).

## 27.5 List of states

Examples of the list of states and data (with explanatory notes) are shown in sections 27.5.1 to 27.5.3 below, for a basin of attraction field of a small RBN<sup>3</sup> defined in figure 27.7. The list of states can be very lengthy so a small network,  $n=5$ , 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 27.3. 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 27.4. Compare these lists with the basin of attraction field and its nodes shown in figure 27.7.

### 27.5.1 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 27.3, or **S** to exclude the extra data. The first column is the state bit-string (or value-string for multi-value). The next three columns 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 26.2) 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.

<sup>3</sup>Note that in an RBN compression does not apply (section 26.2).

<u>data details</u>	<u>attractors-only</u>	<u>decode</u>
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)

## 27.5.2 The list of states with basin data

This example shows the same list as in section 27.5.1 together with basin data. Enter **1s** in section 27.3,

<u>data</u>	<u>decode</u>
v2k3 rcode(dec)110 (hex)6e	
1d size=5 ld=0.625 ld-r=0.75 P=0.625	
z1=0.75 zr=0.625 Z=0.75 C=0/3	... network parameters, described in section 27.4.1 above
ty. at no (p) s % g gd ml mp	... reminder of data order, key in section 27.4.3
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

### 27.5.3 The list of states with basin *and* tree data

This example shows the same (complete) list as in sections 27.5.1 and 27.5.2 together with basin data *and* tree data (decode notes are omitted). Enter **2s** in section 27.3.

```
v2k3 rcode(dec)110 (hex)6e
ld 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 27.4.1 — 27.4.5.

---

# Chapter 28

## Mutation of attractor basins

*not in TFO-mode.*

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/value in a rule-table or just one wire move in a RBN/DDN. 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/DDN as well as CA, rulemix and  $k$ -mix, and to all the rule types, rcode, kcode and tcode.

Mutants of single basins can be grouped on one screen (section 25.2.3) — 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 top-right mutation prompt is 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 sup-trees with very low in-degree, suitable for encryption [47].

**Post-P** ... (*value-range  $v=2$  only*) for 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 random tcode.
- kcode-k** ... for random kcode.
- kcode-r** ... for random rcode.
- none-n** ... no mutation — 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. Add **w** (**+w**), for example **cw**, **sw**, **bw** or **lw**, to first set the wiring mutation, then the wiring and rule can 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 25.2.3).
- set bits/values-b** ... to set the number of bits/values to flip in rcode (section 28.4).
- single bit/value-1** ... (*the default*) to flip one bit/value at random (section 28.4).

For a rulemix (including a *k*-mix), these option generally apply at all cell locations simultaneously 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 local CA will be re-assigned as a randomly wired (nonlocal) network if wiring mutation is chosen, so compression in basins (section 26.2) 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:**

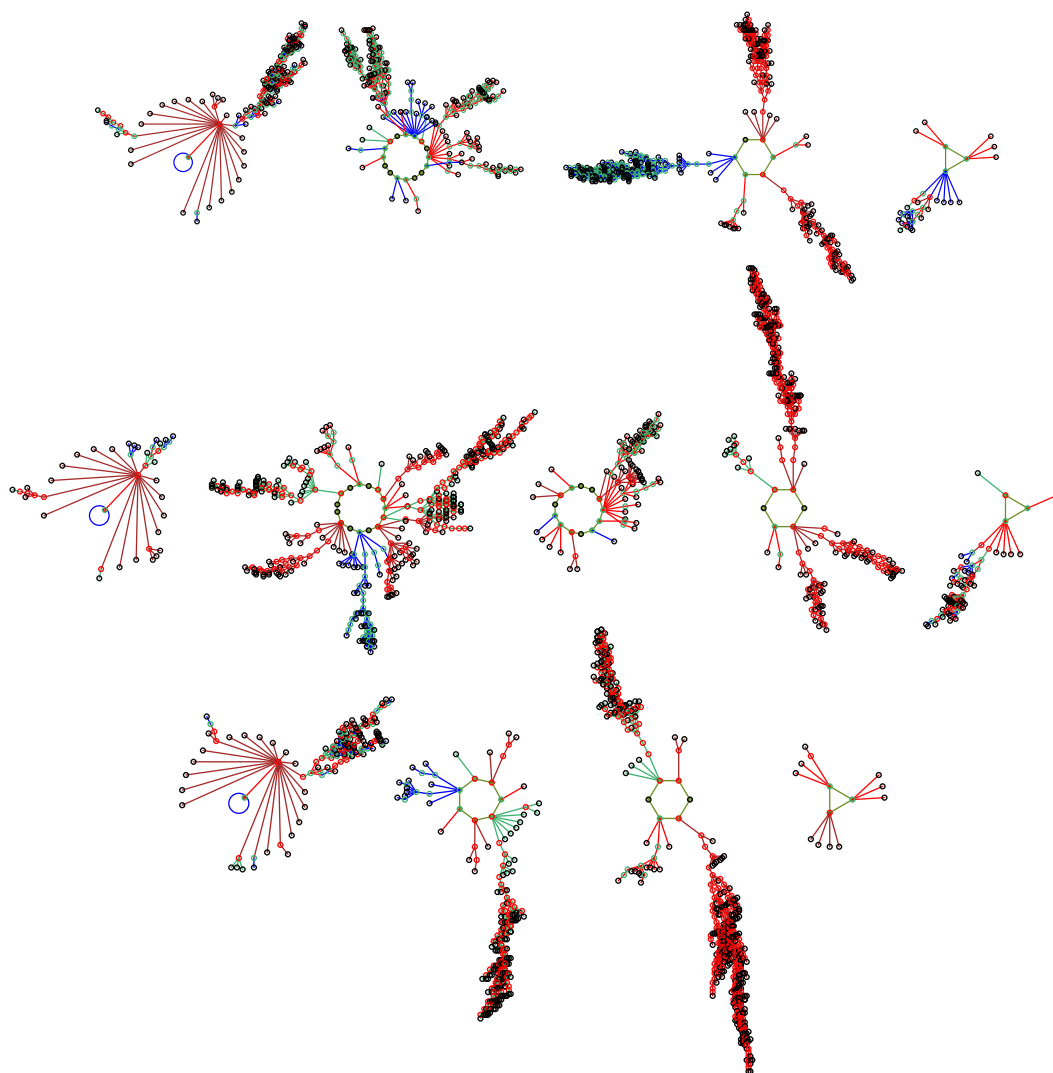


Figure 28.1: Starting with CA  $v2k3$  rule (dec)110,  $n=10$ , as shown in figure 26.4, three mutant basin fields are shown where one randomly chosen wire was moved at random to a new position. The actual moves were as follows: Top: cell 9, 098→398. Center: cell 3, 432→452. Bottom: cell 5, 654→634.

**local-L** ... to restore local wiring (for 1d, 2d or 3d). For 1d and a single rule, compression in basins (section 26.2) will be restored, and the program will revert to 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**.

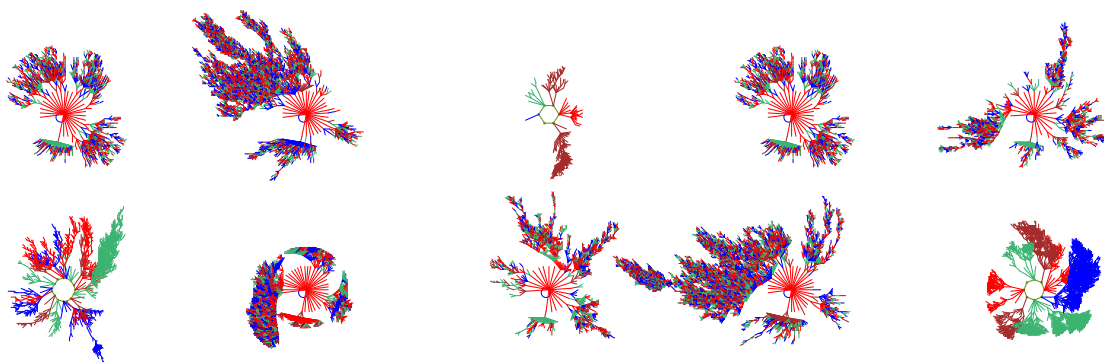


Figure 28.2: Mutant basins of attraction shown on one screen — selected with *mutants-M* in section 25.2.3. The original basin *Top left*: is a CA *v4k2*,  $n=7$ , `kcode(hex)027e60`, `seed(hex)cdca` (as in figure 28.5 but without compression). The other basins are one wire move mutants from the original. As there are many possible single wire moves, this exact set of mutants would be hard to reproduce.

---

## 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, successive mutants can be generated automatically and grouped on one screen (pausing at each full screen) if *mutants-M* is selected in section 25.2.3. Otherwise there would be a pause with the “attractor basin complete” prompt (section 30.4) 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 16.2). For a *rulemix* or *k-mix*, the rule-number at each cell is decreased by one.
2. A succession of random single bit/value flips. For a *rulemix* 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 — from the original string at each flip. For a *rulemix* 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:**

The options, and the wording subject to the context, are described below,



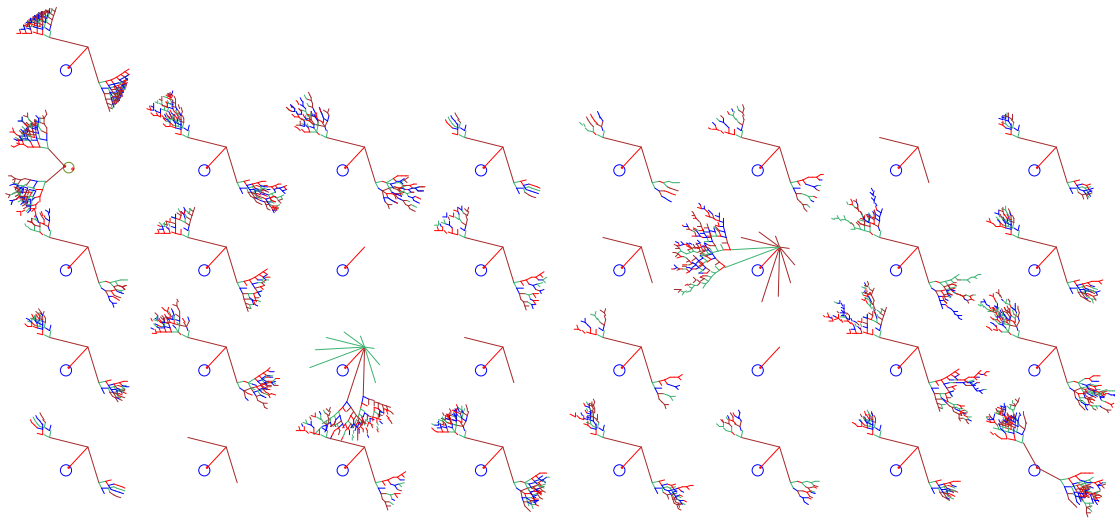


Figure 28.3: 32 one-bit mutant basins of attraction shown on one screen selected with *mutants-M* in section 25.2.3. The original rule *Top left*: is a 1d CA *v3k3*,  $n=8$ , rcode 60, seed all 0s. where all states belong to one very regular basin. The rule was first transformed to its equivalent  $k=5$  rcode (hex)f00ff00f, 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.

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 active) otherwise ... **or tcode...**, or neither, appears.

**kcodeno-5** ... (if kcode active and within decimal limits) for a countdown from the initial decimal kcode number.

**tcodeno-5** ... (if tcode active and within decimal limits) for a countdown from the initial decimal tcode number.

**kcodeflip-4** ... (if kcode active) for a random bit/value flip in kcode.

**tcodeflip-4** ... (if tcode active) for a random bit/value flip in tcode.

**rcodeno-3** ... (if rcode 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* was selected in section 25.2.3).

**bit/value flip from left:** ... (not for *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 from the original rule.

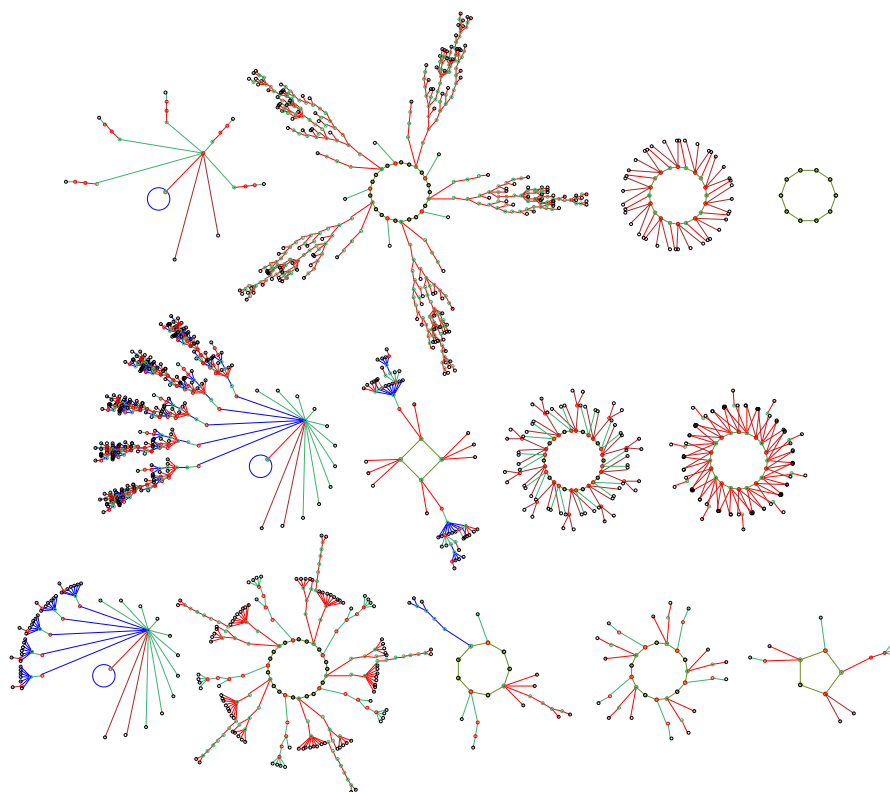


Figure 28.4: Mutation of the basin of attraction field by one bit-flips. The  $v2k3$  rcode (dec)110 was first transformed to the equivalent  $k5$  rcode (hex)3cfc3cfc (section 18.7.1). The CA,  $n=10$  (basin field shown in figure 26.3), was then mutated in sequence by a single bit from the left (non-cumulatively). Three mutant basin fields are shown (from the top) for the 5th, 8th and 14th bit-flip. The three 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 example for  $v2k3$  rcode,

```

rcode: specify start - end position of bit/value flip (countdown)
start position (def 31):      end position (def 0):

```

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.

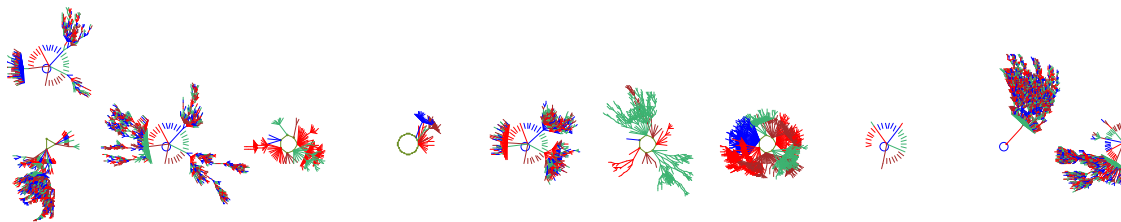


Figure 28.5: Mutant basins of attraction shown on one screen — selected with *mutants-M* in section 25.2.3. The original basin Top left: is a CA  $v4k2$ ,  $n=7$ , kcode (hex)027e60, seed (hex)cdca (as in figure 28.2 but without CA compression active). Lower row: 10 one-value (non-cumulative) mutant 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 26.3), and equivalent trees (and subtrees) (section 26.2.3), are suppressed.

For a rulemix the mutation applies to the rule at each cell simultaneously.

Note that finer mutations can be made to rcode if first transformed to an equivalent rcode with greater  $k$  (section 18.7.1). 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 25.2.3). 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 give 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,

$$\begin{aligned} \text{single rule } \dots & T=S \\ \text{rulemix } \dots & T=S \times n \\ k\text{-mix } \dots & T=\sum_{i=0}^{n-1} S_i \end{aligned}$$

$T$  appears in the prompt and is the basis for the percentage setting. Examples of the 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<sup>1</sup> 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,

**don't pause after mutant-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**.

<sup>1</sup>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

*not in TFO-mode.*

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 behavior of large networks. For a single basin, its also possible to exclude subtrees altogether and show just the “bare” attractor – usefull for very large 2d networks as in figures 29.2 and 29.3.

One of two different reverse algorithms that compute pre-images directly are applied automatically by default for either local or nonlocal wiring. However, the nonlocal algorithm can be applied to 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<sup>1</sup>. **s/+s** toggles the current setting for showing backward space-time patterns (section 24.10). Enter **S** to abandon further options in this chapter and proceed with the options for just space-time patterns (chapter 31).

---

<sup>1</sup>Select a subtree if “neutral order components” are to be generated (section 29.10).

## 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<sup>2</sup>, 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 before the basin is drawn, described in section 29.4. Although the upper size limit for the network is very large (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, against which the state at each new time-step is checked looking for a repeat, which would define the attractor cycle. In previous versions of DDLab, the maximum number of steps in the history array (the history limit) was 65535 (unsigned short int), but is now increased to 4294967295 (unsigned int), so the history limit is unlikely to be an issue<sup>3</sup>.

### 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:
```

Enter **return** to continue looking, or **q** to give up — the top-right attractor basin complete prompt will appear (section 30.4).

<sup>2</sup>However, enter **return** for zero forward steps if “neutral order components” are to be generated (section 29.10).

<sup>3</sup>If the number of iterations exceeds the history limit, no more time-steps will be added, but the checks of new time-steps against the history array continue — for a large state-space and chaotic rules (especially chain-rules section 16.11) the attractor may therefore not be found.

---

## 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 nonlocal) 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 subtree with nonlocal wiring*

**subtree (nonlocal) 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 nonlocal wiring*

**single basin (local) n=14, nonlocal-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, nonlocal-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), (nonlocal)** ... refers to the wiring — whether local as in 1d CA, or nonlocal as in RBN, DDN, or for any other network with nonlocal wiring, including 2d and 3d CA<sup>4</sup>. Local/nonlocal is toggled with **x** (section 29.4.2).

### 29.4.2 Final attractor basin prompt — options summary

Enter **return** to generate the attractor basin with the default reverse algorithm for local or nonlocal wiring. Other options are summarized below, and described in detail in the rest of this chapter,

---

<sup>4</sup>2d and 3d CA “local” wiring is treated as nonlocal in DDLab’s algorithms.

*options ... what they mean*

- nonlocal-x** ... for 1d local wiring, enter **x** to redefine the wiring as nonlocal and use the nonlocal algorithm.
- local-x** ... for nonlocal wiring (except mixed- $k$ ), enter **x** to restore 1d local wiring (if changed to nonlocal above), or to rewire any nonlocally wired network with 1d local wiring — the nonlocal 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 nonlocal wiring (section 29.6.2).
- +pause-2** ... (for nonlocal 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 nonlocal 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 subtree 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 subtree will stop generating further pre-images when the number is reached. Enter **return** not to impose a limit.



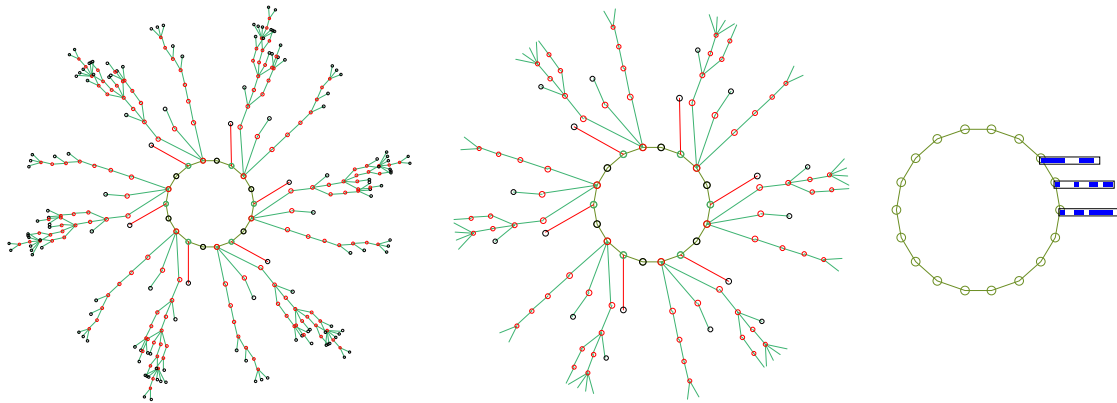


Figure 29.1: Limiting the number of backward steps from the attractor in the trees of a single basins. *Left*: the complete basin. *Center*: trees limited to 5 backward steps. *Right*: trees limited to zero backward steps, and non-equivalent nodes shown as bit-patterns.  $v2k3$  rcode(dec)110,  $n=12$ , seed (hex)08e0.

### 29.5.1 The bare attractor — limit backward steps to zero

For single basins the number of backward steps can be set at zero. Enter **0** at the prompt in section 29.5 to display just the bare attractor. Because the data is derived from forward iteration – reverse algorithms do not come into play – the graphic image of attractors for very large networks, including 2d, can be generated, as in figures 29.2 and 29.3.

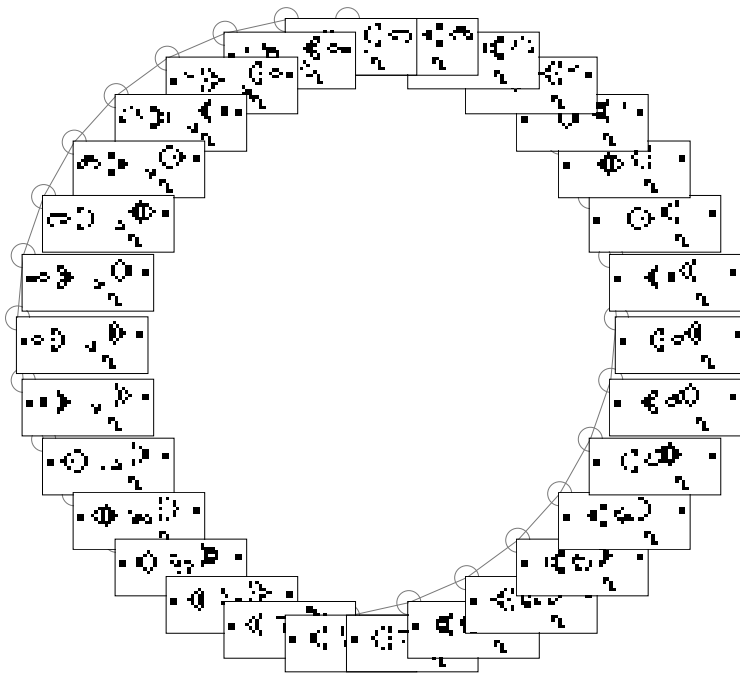
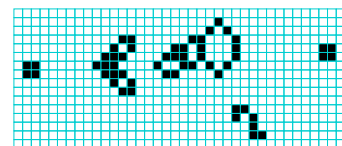


Figure 29.2: *Left* Gosper's glider-gun from the game-of-Life [10] shown as an attractor with period 30, with backward time-steps limited to zero. *Below* The initial state  $38 \times 16$  including an eater to consume emergent gliders.



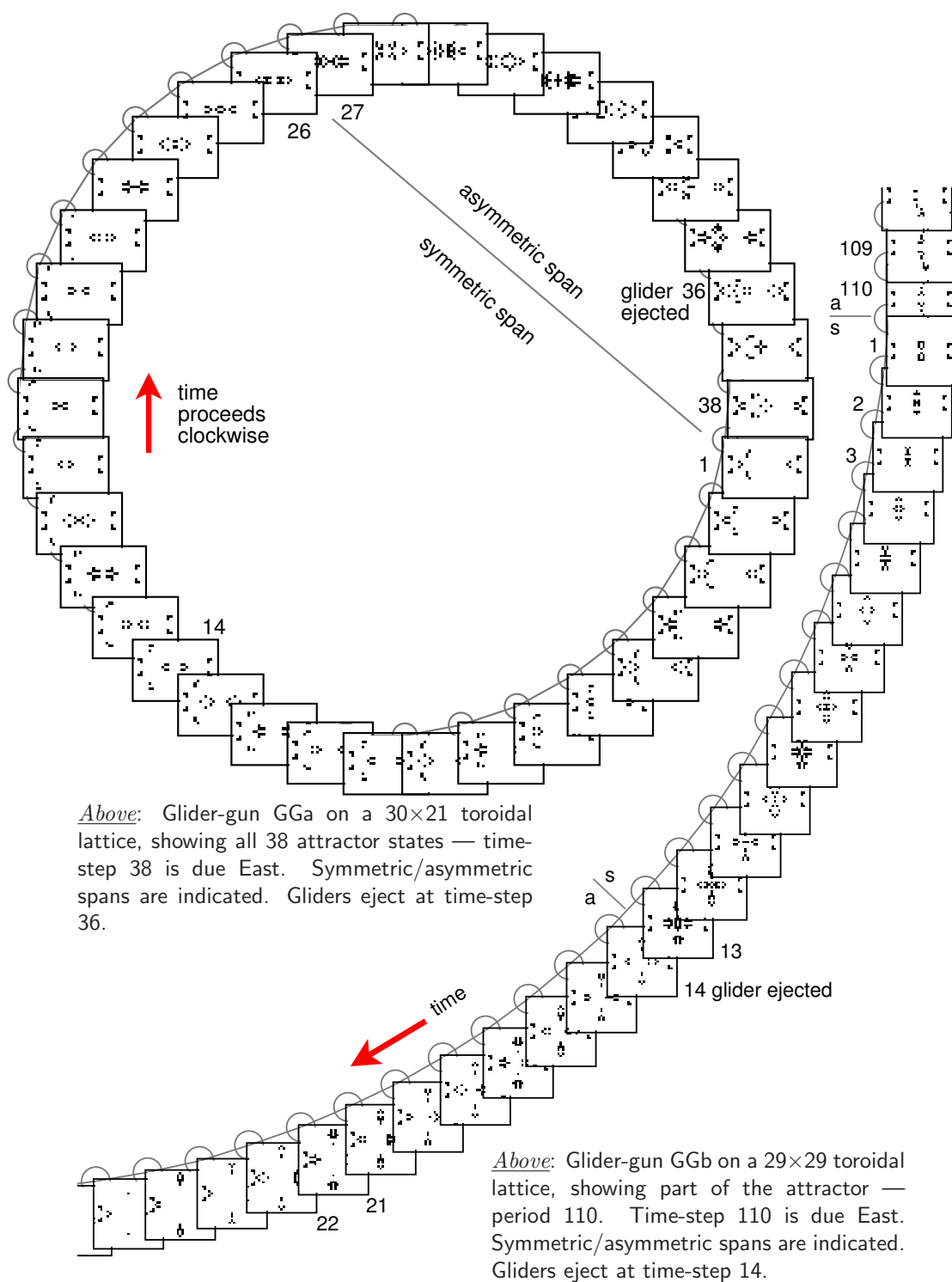


Figure 29.3: Glider-guns, GGa and GGb, from the X-rule [13] shown as periodic attractors where colliding gliders self-destruct. Backward time-steps were limited to zero.

## 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 [31, 32, 37] — for either local or nonlocal wiring. The local algorithm is for homogeneous- $k$  networks with 1d CA (nearest neighbor) wiring, but with possibly mixed rules. The nonlocal algorithm is for any other type of wiring (RBN, DDN, mixed- $k$ , 2d or 3d networks). The nonlocal 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 nonlocal wiring algorithm (section 29.6.2).

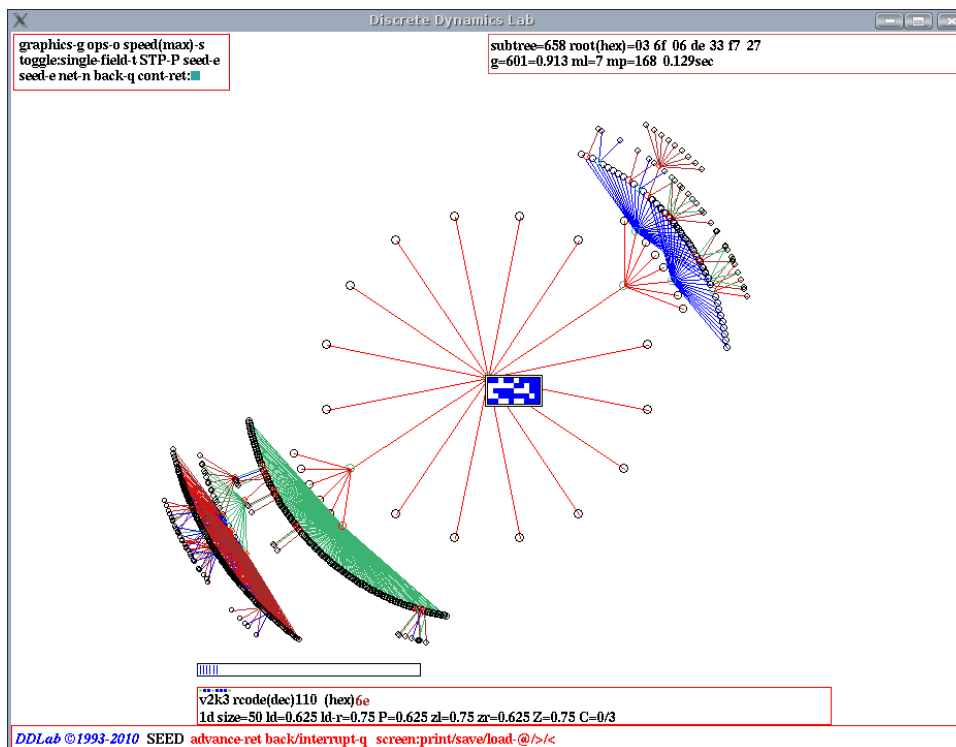


Figure 29.4: A 1d local CA subtree, showing the partial pre-image stack,  $v2k6$  rcode(dec)110,  $n=50$ . The partial pre-image stack is represented by a horizontal bar extending from left to right within a rectangle near the bottom of the screen — its length (in pixels) shows the varying size as the partial pre-image stack grows and shrinks, leaving a trace of its maximum extent (shown in this example). It may be necessary to slow computation (sections 24.11, 30.4) to see the bar.

### 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 29.4).

### 29.6.2 Partial pre-image stack, nonlocal wiring

For nonlocal wiring, enter **1** or **2** in section 29.4 for a view of the inner workings of the nonlocal reverse algorithm. A histogram will appear in the lower half of the screen with  $n$  columns (figure 29.4) 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 presented,

**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:** *(for example)*

*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 wild-cards.

**abandon pause-p** ... to abandon 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.

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.

### 29.6.3 Reorder to optimize the nonlocal reverse algorithm

The algorithm to reorder cells for more efficient computation is implemented 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* implements the reverse algorithm from left to right.

The nonlocal reverse algorithm [32, 35] 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 29.5).

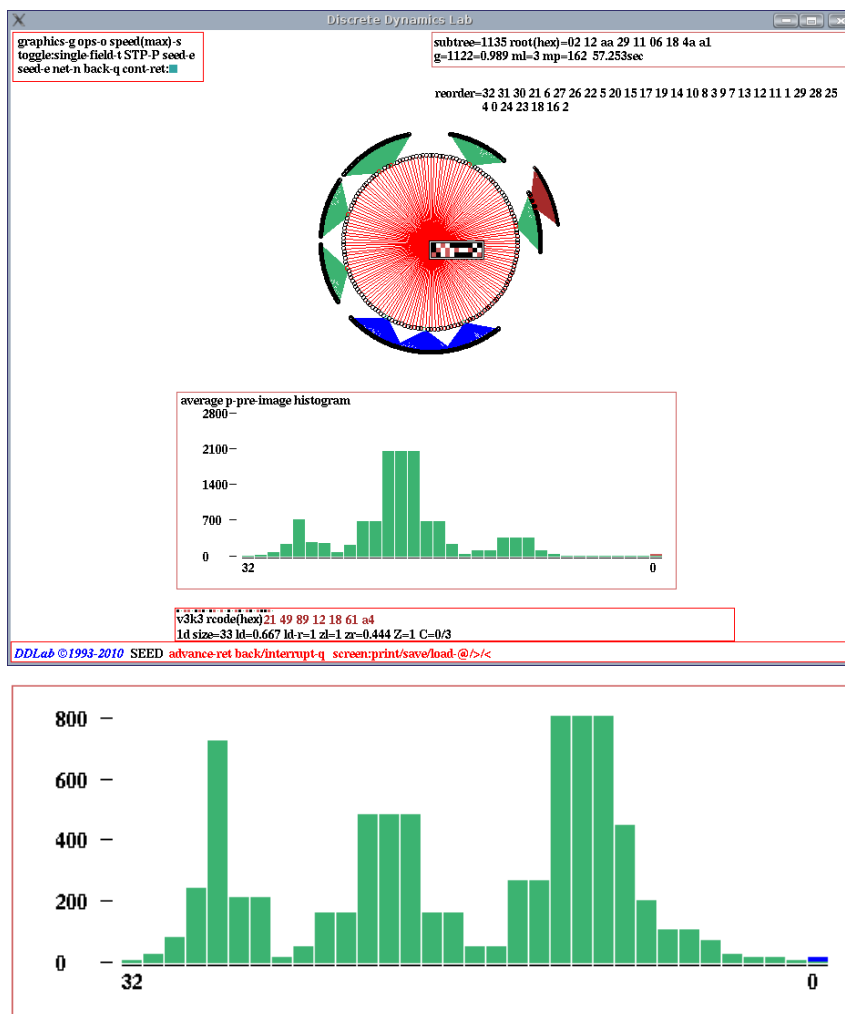


Figure 29.5: A 1d nonlocal CA subtree ( $v3k3$ ,  $n=33$  — defined below). The changing histogram shows the varying size of the partial pre-image stack for each *next* cell. The final bar height represents the number of pre-images of a particular state. The numbers below the top-right window show how cells were reordered to optimize the nonlocal 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 `v3n33a1g.wso`, 2 steps forward from the initial state (hex) 010a154814556a4201 before running backwards.

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 successive cells (if possible), to allow conflicts to happen early. For fully random wiring, this reduces the growth of the partial pre-image stack considerably compared to a left to right (or some arbitrary) order. A left to right order is of course very efficient for 1d CA wiring. Memory load can be

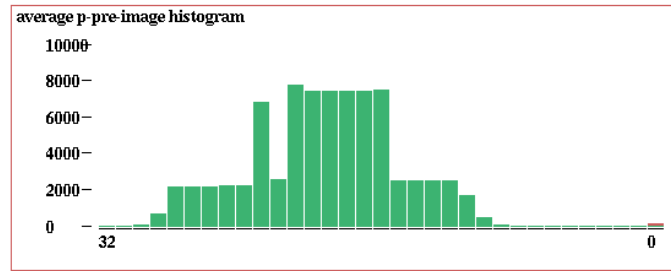


Figure 29.6: The final partial pre-image histogram, with the reordering algorithm suppressed, for the same network as in figure 29.5 *Top*. An approximate comparison of the time  $t$ , and the maximum average stack height  $h$ , with the reordering algorithm active is as follows, active:  $t=57\text{sec}$ ,  $h=2000$ , inactive:  $t=5\text{min } 51\text{sec}$ ,  $h=8000$ .

reduced and computation speed increased. The new order (if applied) is shown below the top-right prompt window (figure 29.5). The reordering reduced the time (and memory) to complete this subtree by a factor of about 6 (figure 29.6).

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, 2.19). The following exhaustive pairs prompt (below the top-right) is presented, and options summarized below.

**exhaustive pairs: compute-ret, and save-s, and prt-p/+p: (prt-p/+p not for DOS)**

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 29.7). The exhaustive algorithm also applies to asynchronous updating (section 29.9).

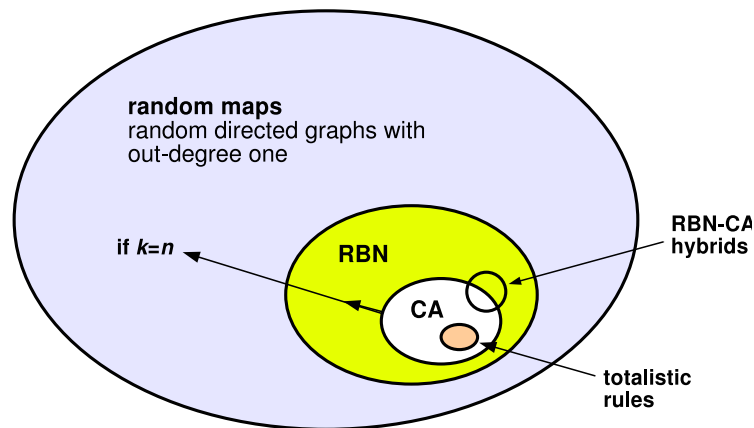


Figure 29.7: CA, DDN/RBN, and random maps, are systems that form a hierarchy of subsets, with random maps the most general. This is reflected by the reverse algorithms, for (a) 1d local wiring, (b) nonlocal wiring but including (a), and (c) exhaustive testing which includes (a) and (b) as well as random maps and asynchronous updating.

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 $v^n \approx$ millions
2	28	268435455 $\approx$ 268.4
3	18	387420488 $\approx$ 387.4
4	14	268435455 $\approx$ 268.4
5	12	244140624 $\approx$ 244.1
6	11	362797055 $\approx$ 262.8
7	10	282475248 $\approx$ 282.5
8	9	134217727 $\approx$ 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 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 (sections 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 (with the exhaustive algorithm, section 29.7) 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:**  
(**Hamx** *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.
- set Ham1-(def)** ... enter **return** to create a random map with the current Hamming distance bias, shown as **Hamx**, 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.

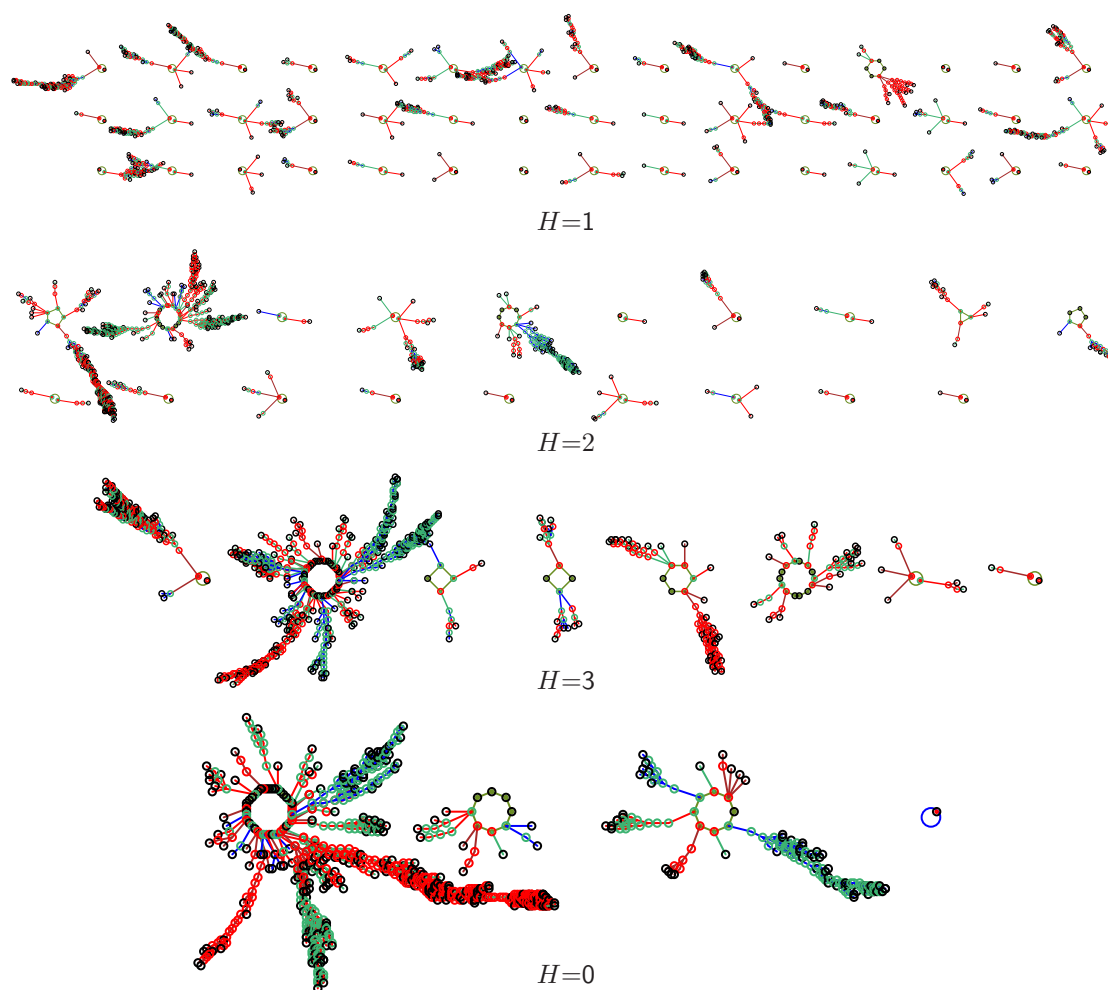


Figure 29.8: Typical random map graphs (basin of attraction fields) with different Hamming bias  $H$ . From the top,  $H=1$ ,  $H=2$ ,  $H=3$ , and without bias  $H=0$ , so fully random.  $H=1$  is the most fragmented,  $H=0$  tends to have a characteristic giant component[35, ch.3].

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 with converging trajectories leading to attractors, though such a system is very probably 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<sup>5</sup> such as the following will be displayed,

```
file n=7 != network n=8 (wrong file) cont-ret: (values shown are examples)
```

The default filename for a random map (or exhaustive pairs) file is `myexh_vx.exh` where  $x$  is the current value-range  $v$ . The (binary) encoding is described in section 35.2 (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 29.8 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 fragmented,  $H=0$  tends to have a characteristic giant component[35, 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.

### 29.8.3 Random map data

In addition to the normal data shown in a top-right window when an attractor basin 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[35, ch.3]. This example (from figure 29.8  $H=2$ ) 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)
```

---

<sup>5</sup>A mismatch in  $v$  will usually be detected because the file size will be incorrect, but this will be reported as a mismatch in  $n$ .

## 29.9 Sequential updating

with the exhaustive algorithm (section 29.7) so  $v^n$  within table 29.1

By default, network updating is synchronous, in parallel. Alternatively, the updating can be sequential<sup>6</sup>. 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 for sequential updating. with the following top-right prompt,

```

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 summarized 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 10.2).  
**<-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).

### 29.9.1 Random order

Enter **r** in section 29.9 (or in section 31.4.2 for space-time patterns<sup>7</sup>) 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:

```

<sup>6</sup>This section is also relevant to sequential updating for space-time patterns (section 31.4.2), but in that case the  $v^n$  limit in table 29.1 is irrelevant.

<sup>7</sup>For space-time patterns (section 31.4.2) — for large networks there are two extra options: **more-m** to see the next window of the update order, and **jump-j** to jump to a selected update-tag.

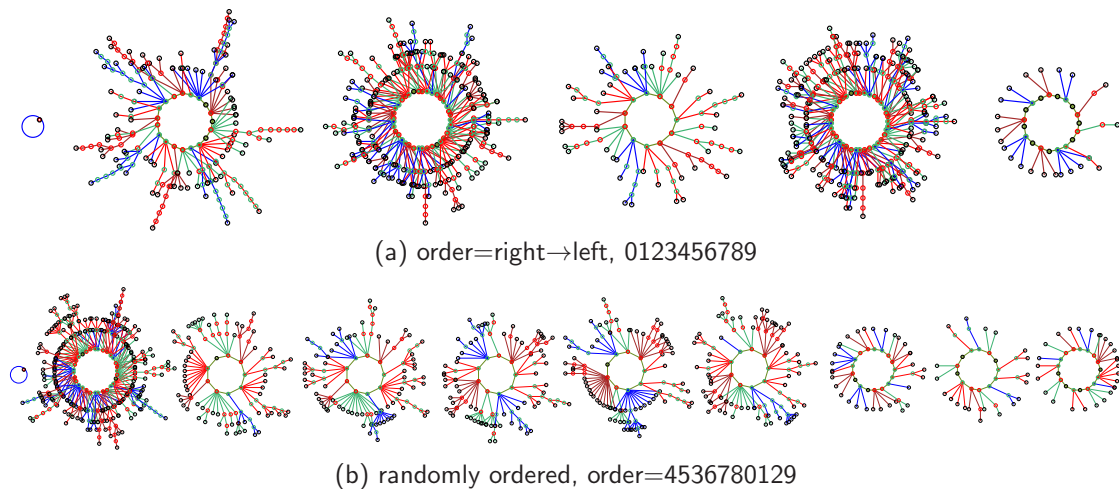


Figure 29.9: A basin of attraction field for a 1d CA with sequential updating,  $v2k3$  rule (dec)110,  $n=10$ . (a) sequential left to right, (b) randomly sequential. The order is shown in the lower rule window.

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 number 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 (section 35.3).

**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 successive 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 complete 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 - 01323456789ab - 0 (this is the right to left order)
1 - 01323456789ba - 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<sup>8</sup> 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 summarized 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<sup>9</sup> 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 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 29.10: The rule window for sequential updating. This example is for  $v2k3$  rcode(dec)110,  $n=10$ , and a sequential order as in figure 29.9(b).

---

## 29.10 Neutral order components

*PBC,  $v=2$ ,  $n \leq 12$  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

<sup>8</sup>Note that some of the sub-options in section 29.9.4 may be irrelevant in the context of the order seed.

<sup>9</sup>This may take some time for larger networks, as noted in section 29.7).





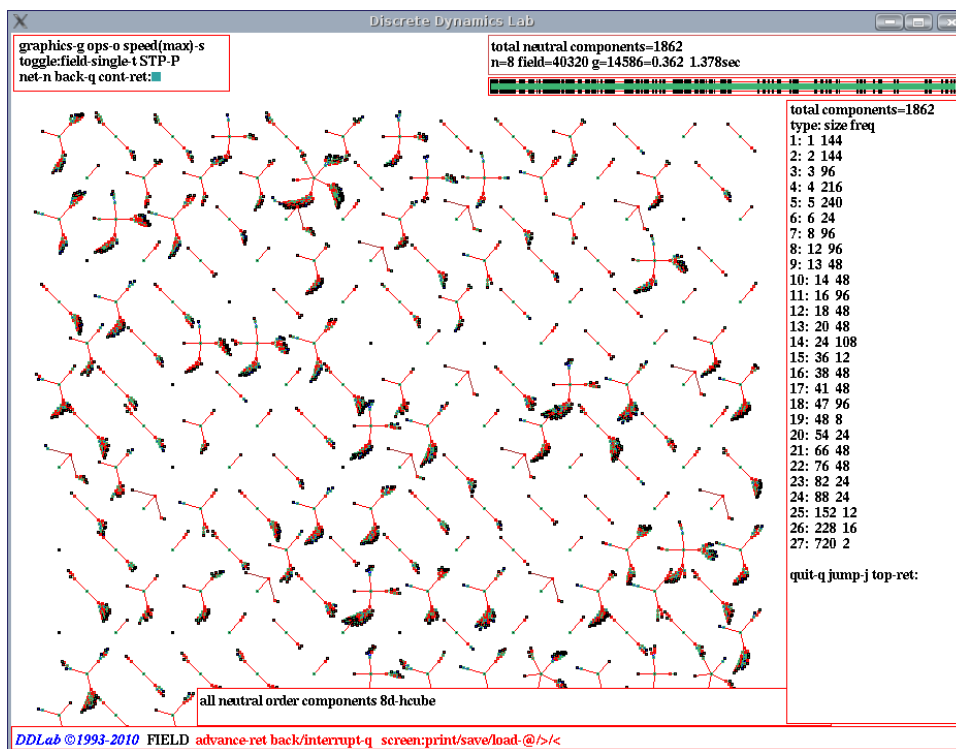


Figure 29.12: The neutral field, the subtrees making up sequence space — the first 175 or so, out of a total of 1862, are shown — computed and drawn in an analogous way to a basin of attraction field for network dynamics. A list of component sizes and the frequency of different sizes is shown in a window 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

### 29.10.1 Neutral subtree

To 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,

```
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 in 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, (section 12.2). As the subtree is done, 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<sup>11</sup> (figure 29.12). 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 27.1.2.

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 29.12, 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)
```

The prompts on the bottom line are explained below,

options ... what they mean

**quit-q ...** Enter **q** to backtrack and show a summary of the neutral field in a top-right window, as in figure 29.12, 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.

<sup>11</sup>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.

**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 25.2.8.

---

# Chapter 30

## Drawing attractor basins, and changes on-the-fly

*not in TFO-mode.*

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 27.1.2, 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 27.2) indicates how much of state-space has been used up so far. A bar that over-shoots 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.

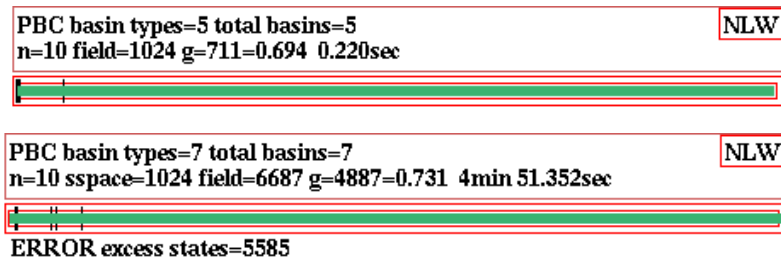


Figure 30.1: Examples of the basin of attraction field progress bar and the data window for an RBN. *Top:* For a normal run without error. *Bottom:* An example of an **ERROR** excess states=, where the size of the field exceeds state-space. This is caused if wiring was changed, or a tree or basin skipped, during a pause (section 30.2). The progress bar has also overshoot its scale on the right.

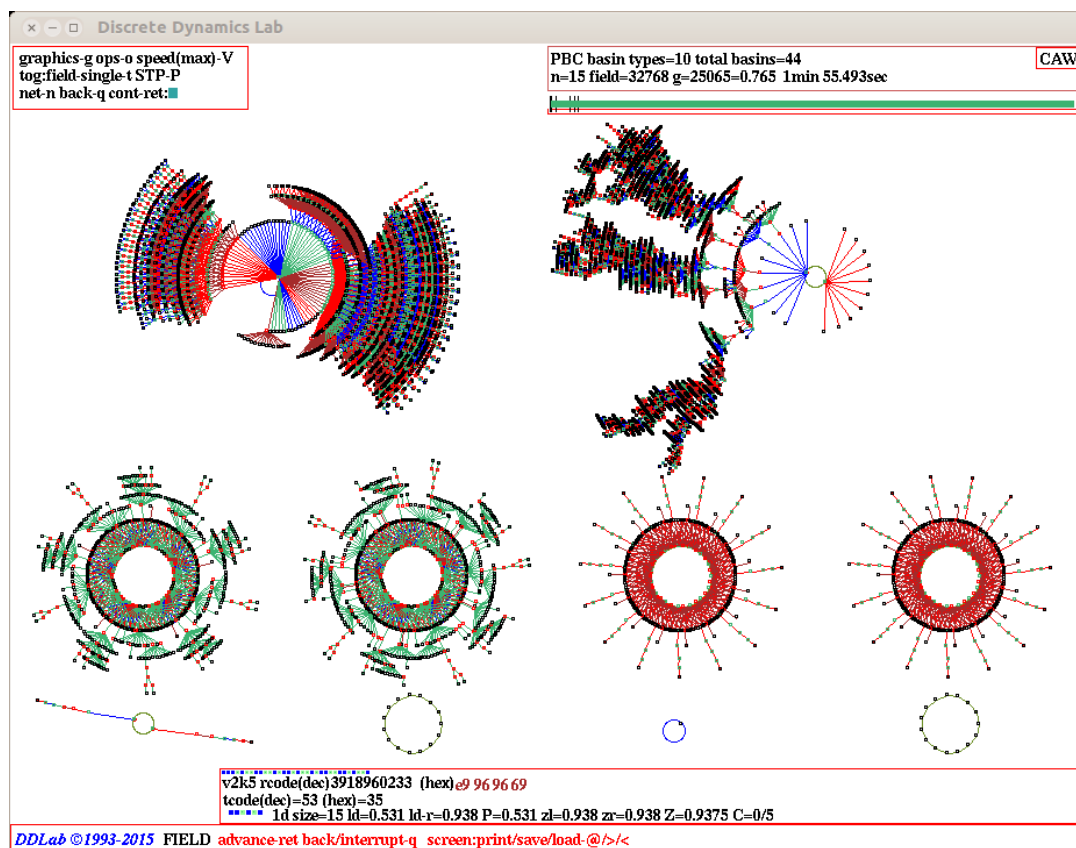


Figure 30.2: The DDLab screen showing a basin of attraction field. This example is for a 1d CA,  $n=15$ ,  $v2k5$  tcode (dec)53. To achieve this layout, a pause was selected after each basin in section 27.1.2, and the position and spacing of basins were amended as described in section 25.3. Various 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 27.2).
- below top-right:* ... the progress bar (section 30.1).
- bottom-center:* ... 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 while being drawn. A top right notice/prompt will appear with content depending on where the reverse algorithm was interrupted:

- (a): while computing the pre-image fan.
- (b): between pre-image fans but before the tree is complete.
- (c): while still looking for the attractor cycle.

early exit - in pre-image fan (2199 found) ... (a)  
 level 31, incomplete tree=5063 ... (b)  
 still looking for attractor cycle (1845 iterations) exit-q, cont-ret: ... (c)

For (a) and (b) a second prompt line is presented, which depends on whether the attractor basin is a subtree, single basin or a basin of attraction field,

*for a subtree*

**options-o stopsubtree-q speed-V cont-ret:**

*for a single basin*

**next-tree-n options-o stopbasin-q speed-V cont-ret:**

*for a basin of attraction field*

**next-tree/basin-n options-o stopfield-q speed-V 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*)

These options are summarized 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-V** ... to slow down drawing attractor basins and backwards space-time patterns or revert to full speed. See section 24.11, which also lists other stages in DDLab where slow motion can be invoked, including on-the-fly.
- stopfield-q** ... (or **stopbasin-q**, or **stopsubtree-q**) 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 implementation 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 26.2.2). If compression (section 26.2) was not deactivated in section 26.2.1, 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 27.2). 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 27.2), preceded by the words “EARLY EXIT” to indicate that the data applies to an incomplete attractor basin. For example,

```
EARLY EXIT PBC 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 and backwards space-time patterns on-the-fly — the following reminder is shown in the bottom title bar (section 5.5), depending on the type of system selected in section 29.4,

*for nonlocal 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 nonlocal wiring reverse algorithm (section 29.6.2). The option **matrix-m** does not apply for neutral order components (section 29.10). The key hits are described below,

*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 on and off (section 24.10), **#** 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 shown as bit/value patterns (section 26.3).

**ppstack:tog-P** ... (for the nonlocal reverse algorithm only) enter **P** to toggle showing the partial pre-image histogram (section 29.6.2). The histogram will start immediately and at the same time 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. Slow motion can also be invoked at various other stages in DDLab, listed in section 24.11.

These changes take effect as soon as the key is pressed, without **return**, but basins may be drawn too fast to hit a key in time. To allow for this, computation may be slowed at various other stages in DDLab, listed in section 24.11.

## 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)-V** (or **speed(slow)-V** if not currently *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)-V** (or **speed(slow)-s** if not currently *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 summarized below,



*options ... what they mean*

- 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-V** ... the current speed status is either (**max**) or (**slow**). Enter **V** to slow down drawing attractor basins and backwards space-time patterns or revert to full speed. See section 24.11, which also lists other stages in DDLab where slow motion can be invoked, including on-the-fly.
- 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.
- STP-P** ... to toggle between showing and not showing “backwards” space-time patterns (section 24.10).
- 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 many network architecture options described in chapter 17, including viewing, revising and filing, the Derrida plot (chapter 22), and learning (chapter 34). Following this, DDLab reverts to 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 only. A top-right prompt similar to the following is presented,

```
rule:save/load/revise/rnd/trans-s/l/v/r/t net-n
valueflip-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 (for a single basin or subtree)
back-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 generally 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 allv-b Z:higher/lower-Z/z canal-C (Z: for single rule networks only)
```

The meaning of the options are summarized below,

*options ... what they mean*

**save/load-s/l** ... enter **s** or **l** to save or load the rule ( 35.3). For mixed rule networks this applies to the rule at cell index 0 only.

**revise-v** ... to review/revise the rule in a lower right window as described in chapter 16. 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)

**rnd-r** ... to reset a new random rule, or all rules at random in a rulemix.

**trans-t** ... to transform the rule as described in chapter 18. 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)

**net-n** ... for the many network architecture options described in chapter 17, including viewing, revising, transforming, 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 rulemix.

**allv-b** ... to flip outputs of uniform neighborhoods to the neighborhood value, all0s→0, all1s→1, all2s→2 etc. — in a single rule, or for all rules in a rulemix.

**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 18.6), or for all rules in a rulemix (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 summarized below,

options ... what they mean

**rev-e** ... to revise the seed (also available in section 30.4). A seed prompt (section 21.1) is presented in a lower center window.

**rnd-R** ... for a new random seed. The preset density-bias of the seed (section 21.3) is respected.

**blk-k** ... for a random central block of cells. The rest can be kept as is, or set to zero. The block size was originally specified in section 21.3 but can be changed here. The following top-right prompt is presented,

**rnd block: change size (now 4), rest: keep-k, all 0s-(def):**

If a new block size is entered the prompt reappears showing the new size. Enter **k** to keep the rest of the seed, **return** for the new block on a zero background. The preset density-bias (section 21.3.2) of the block is are respected.

**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 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** ... to save the original seed, or the current state — the latest state in the attractor basin. — the following top-right prompt is presented,

**save seed: original-o, current-c:**

The seed is saved as a `.eed` file (section 35.3).

**load-L** ... to load a new seed, a `.eed` file (section 35.3), subject to the constraints listed in section 21.7.

---

# Chapter 31

## Output parameters for space-time patterns

The output parameters for space-time pattern allow various default settings to be revised before space-time patterns start. Many of these settings, and some extra ones, can also be invoked on-the-fly — while space-time patterns are running (chapter 32), and from the interrupt window (section 32.16). Space-time patterns (running forward) apply in the following circumstances:

- in TFO-mode (section 6.2.1): the first output parameter appears after setting the seed (chapter 21).
- in SEED-mode (section 6.2.2): the first output parameter prompt appears ...
  - if *space-time patterns only-s* was selected at the “first output parameter prompt for attractor basins” (section 24.1),
  - or alternatively at a final stage, if *space-time patterns only-S* was selected at the “subtree or single basin” prompt (section 29.1).

Its possible to jump directly to a particular category of options, namely *updating*, *entropy*, *damage*, *attractors*, and *skeletons*, apart from the first set of *start/misc* miscellaneous options. Although there are many output parameter options and sub-options relating to space-time patterns, all have defaults — any remaining options can be skipped at any time by entering **d**.

---

### 31.1 The first output parameter prompt for space-time patterns

The first output parameter prompt for space-time patterns is presented in a top-right window. At the same time **space-time parameters** appears in a top-center window.

**accept all space-time defaults-d**  
**revise from:** **start/misc-ret boundaries-b updating-u**  
**entropy-e damage-m attractors-a skeletons-s:** (*skeletons for v=2 only*)

### 31.1.1 Output parameter prompt for space-time patterns — options summary

The output parameter options may be looked at in turn (up to **damage-m**), but they are divided into categories to allow jumping directly to the category where options need to be set. The categories in (section 31.1) are summarised below, together with subcategories where applicable — and described in detail in the rest of this chapter. As soon as a “**revise from**” option is selected, the reminder **accept defaults-d** appears in a top-center window – enter **d** (at any time) to accept all remaining defaults, and skip further output parameter prompts. Enter **q** to backtrack. Options marked with an asterisk (\*) can also be modified on-the-fly (chapter 32) and with a carat (^) from the interrupt/pause prompt (section 32.16).

*categories ... what they mean, and section in this chapter*

- start/misc-ret** ... enter **return** to access all the options in turn, (except attractors-a and skeletons-s) starting with the miscellaneous options 31.2.
- \*31.2.1 ... cell color by neighborhood or value.
  - \*31.2.2 ... plotting the state-space matrix and return map.
  - \*31.2.3 ... the threshold for showing “frozen” cells.
  - \*31.2.4 ... cell-scale.
  - \*31.2.5 ... displaying space-time patterns in other than “native” dimensions — 1d, 2d, or 3d.
  - \*31.2.6 ... sweeping or scrolling space-time patterns.
  - ^\*31.2.7 ... options for pausing or stepping through space-time patterns.
  - ^\*31.2.8 ... options for slowing down space-time patterns.
  - ^31.2.9 ... (*if the files are accessible*) select sequential or random order for loading lists of glider rules on-the-fly.
  - 31.2.10 ... (*TFO-mode only*) inverting the kcode.
- boundaries-b** ... to set null boundary conditions instead of the default periodic boundary conditions (section 31.3).
- updating-u** ... jump directly to options for asynchronous or noisy ways to update the network (section 31.4).
- \*31.4.1 ... probabilistic updating.
  - \*31.4.2 ... sequential updating.
  - \*31.4.3 ... partial order updating.
- entropy-e** ... jump directly to methods for analyzing space-time patterns by input-entropy and pattern density (section 31.5).
- 31.5.1 ... single cell input-entropy and pattern density.
  - \*31.5.2 ... generation size for input-entropy and pattern density - of the moving window of trailing time-steps.
- damage-m** ... jump directly to damage spread from perturbations — the difference between two networks (section 31.6):
- 31.6.2 ... a histogram of damage spread for a sample of initial states.
- attractors-a** ... jump directly (the only way) to find attractors by running forwards and creating a histogram of attractor types (section 31.7).

**skeletons-s** ... (for binary systems,  $v=2$ , only) jump directly (the only way) to find “skeletons” by running forwards and creating a histogram of skeleton types. Skeletons are fuzzy attractors where a predefined proportion of the network has stabilized (section 31.8).

## 31.2 Miscellaneous options — space-time patterns

Enter **return** at the first output parameter prompt (section 31.1) for the first category of miscellaneous (hard to categorize) options, starting with section 31.2.1 below. All the output parameter options in all categories could be accessed in turn if required.

### 31.2.1 Color cells by value or neighborhood

By default each cell is colored according to its value<sup>1</sup> (chapter 7). Alternatively cells may be 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 or 256 (section 6.3.4). The option applies to the full value-range but is mainly intended for binary systems,  $v=2$ . Color by value or neighborhood can be toggled on-the-fly (key **3**, section 32.9.5), but for color by neighborhood from the start the following top-right prompt is presented,

**start/misc: cellcolor: nhood-n value-(def):**

Figures 31.1, 31.2 and 31.3 show the difference between the value and neighborhood display of  $v=2$  space-time patterns, for 1d, 2d and 3d<sup>2</sup> — these were all generated from a singleton seed, a single central 1.

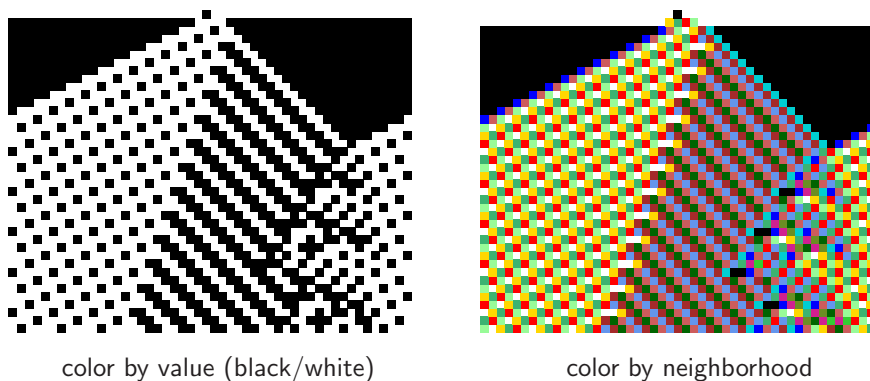


Figure 31.1: 1d space-time pattern, 40 time-steps from a singleton seed, for a CA  $n=50$ ,  $v=2k5$ , rule 8226dc23. Space is across, time is down the page. As the value-range is 2, color by value is simply black and white.

<sup>1</sup>For  $v=2$  the color of 1s starts black but changes between black and blue as various on-the-fly changes are activated (section 32), and black-blue can be toggled (section 32.9.6).

<sup>2</sup>In 3d or 2d+time (section 32.10.2), cells with zero value are not shown — although the neighborhood rule-table index may be positive. This allows a clearer view of the space-time structure which would otherwise be a solid mass.

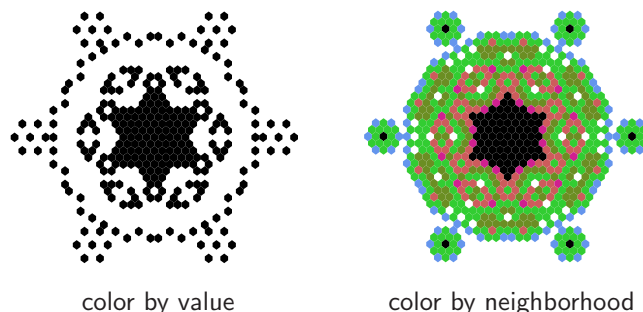


Figure 31.2: 2d space-time pattern. The 19th time-step from a singleton seed, for a CA  $n=38 \times 38$  hex lattice,  $v2k6$  tcode (dec)114.

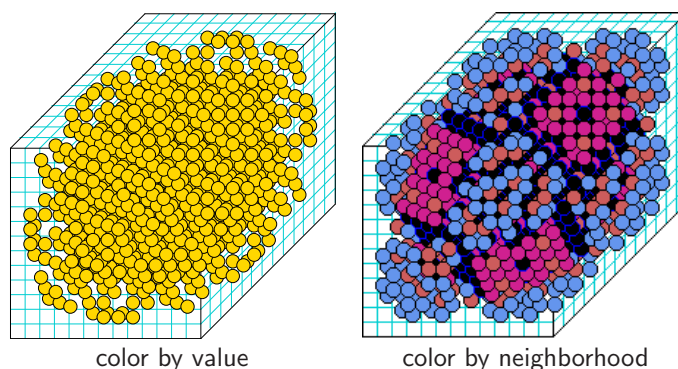


Figure 31.3: 3d space-time pattern. The 90th time-step from a singleton seed, for a CA  $n=13 \times 13 \times 13$ ,  $v2k6$  tcode (dec)114. Note that cells with zero value are not shown in either case to avoid a solid mass.

## 31.2.2 State-space matrix and return map

The “state-space matrix” and “return map by value” in a 2d-phase plane are two methods for representing states as dots on a scatter plot<sup>3</sup>. These plots provide a kind of signature of the dynamics. To speed up the plots considerably, toggle the space-time pattern display off (on-the-fly **S**, section 32.9.8). The following prompt is presented,

**show state-space matrix-y, return map-D:**

Enter **y** to show the state-space matrix, **D** to show the return map by value. Both options can also be toggled on-the-fly (key **y** or **D**, sections 32.12.9 and 31.2.2.2).

### 31.2.2.1 State-space matrix

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 decimal values representing the left half of the bit/value string ( $x$ -axis) against the right half ( $y$ -axis). If  $n$  is odd, the extra

<sup>3</sup>Other types of state-space scatter-plots are the “entropy-density plot” (section 32.12.3), the “return map by density” (32.12.7, and the “Derrida plot” (chapter 22).

bit/value is included on the left, and the grid is a flat rectangle, otherwise the grid is square. Although the state-space matrix will give an output for any network size and value-range, only a limited number,  $X$ , of the most significant bits/values are included from each half of the string to find its decimal value (an unsigned long int).  $X$  for different value-range  $v$  corresponds to table 7.2.

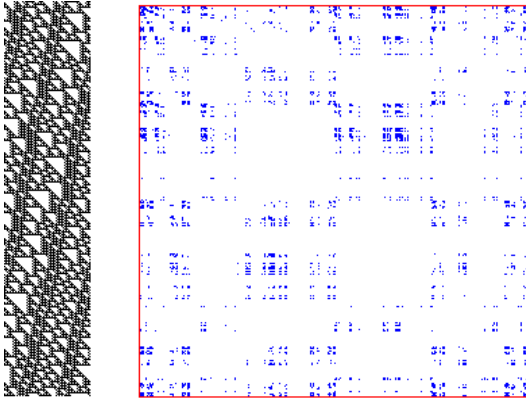


Figure 31.4: The state-space matrix for  $v2k5$  rcode 473ccf18,  $n=62$ , plotting the left half against the right half of each state's bit string — an attractor was reached after about 1200 time-steps.

Left: an example of the space-time pattern.

Right: the state-space matrix.

### 31.2.2.2 Return map by value

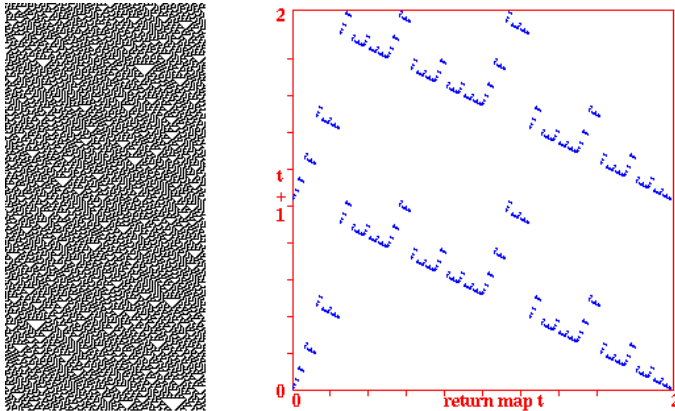


Figure 31.5: The return map for  $v2k3$  rcode 30,  $n=150$ , plotting the state at each time-step, converted into a decimal number 0-2, against its successor, for about 50,000 time-steps.

Left: an example of the space-time pattern.

Right: the return map, note the fractal structure characteristic of chaotic dynamics.

The return map by value in a 2d-phase plane plots each state against its successor. For a binary network, the state (bitstring)  $B_0, B_1, B_2, B_3 \dots B_{n-1}$  is converted into a decimal number  $M(0-2)$  as follows,

$$M = B_0 + B_1/2 + B_2/4 + B_3/8 + \dots + B_{n-1}/2^{n-1}$$

For a mutli-value network, the bits are replaced by values, and each term is divided by  $v-1$  for an equivalent result. As the network is iterated,  $M_t$  at time-step  $t$  (x-axis) is plotted against  $M_{t+1}$  at time-step  $t+1$  (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 31.5.



### 31.2.3 Frozen generation size

Once running, the space-time pattern presentation can be reset on-the-fly to highlight “frozen” cells — that have not changed in the previous  $x$  time-steps, or that fall into preset “frequency bins” (enter **h**, a 4-way toggle, section 32.11.1). 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 20):**

The frozen generation size (and frequency bins) can also be reset on-the-fly (key **H**, section 32.11.2). The current sizes are shown in the “on-the-fly key index”.

### 31.2.4 Cell scale

The cell scale of space-time patterns is set automatically for different network sizes and dimensions, but this can be altered. The following top-right prompt is presented,

**cellscale (now 1 pixel):** *for a 1d network n=150*

Enter the new cell scale in whole pixels. This can also be changed with on-the-fly keys **e** or **c** to expand or contract (section 32.9.11).

### 31.2.5 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 (key **T** a 3-way toggle, in section 32.10.1). One of the following top-right prompt is presented,

**1d network, show STP in 1d-1 2d-2 3d-3 (def 1d):** *(for 1d)*

**2d network, show STP in 1d-1 2d-2 3d-3 (def 2d):** *(for 2d)*

**3d network, show STP in 1d-1 2d-2 3d-3 (def 3d):** *(for 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 32.18. Note that if  $n$  is prime,  $i=n$ , 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.

### 31.2.6 Scrolling 1d space-time patterns

1d space-time patterns will scroll vertically by default on reaching the bottom of the screen. Alternatively the patterns can be made to sweep instead of scroll – restart from the top on reaching the bottom of the screen. For 1d space-time patterns presented in 1d, the following top-right prompt is presented,

**sweep space-time pattern-s, scroll-def:**

Enter **s** to sweep. The two methods can also be toggled on-the-fly<sup>4</sup> (enter **#**, section 32.13.3). If the input-entropy or pattern density is set (section 31.5) these plots will also scroll or sweep.

---

<sup>4</sup>2d presentations of space-time patterns can also be scrolled on-the-fly, either vertically (section 32.10.2), or diagonally (section 32.13.3) — which can also include scrolling the network-graph (section 32.19 and figure 32.15).

### 31.2.7 Pause and step

DDLab offers various methods of scanning space-time patterns, some of which are also available on-the-fly (section 32.13.4), and from the interrupt/pause prompt (section 32.13.4). 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):**

*options ... what they mean*

**step-s** ... to show each next iteration by pressing **return**. This can also be set on-the-fly (key **+**, section 32.13.4).

**screen full-p** ... to pause when the screen is full. This can also be set from the interrupt/pause prompt (key **N**, section 32.13.4).

**time-step** ... enter a number to specify a run of time-steps before a pause. At each pause, enter **x** in the interrupt/pause prompt (section 32.16) for the next run of time-steps, or **return** to stop pausing and continue as normal. The default run equals the “frozen generation” size (default 20 time-steps, section 31.2.3).

### 31.2.8 Speed of iteration

To reduce the speed of space-time pattern iteration, the following top-right prompt is presented,

**speed: max-(def), or slowmotion (try 300):**

Enter **return** for full speed, or a number to slow down — the greater the slower.

Apart from this option, slow motion can also be invoked on-the-fly (sections 32.2 and 32.13.5), from the interrupt prompt (section 32.16.8, and from the “Quit and further options” prompt (section 32.17).

### 31.2.9 Glider rule order

*if a list of complex rules is accessible*

Collections of complex (or “glider”) rules — that produce emergent space-time structures including interacting gliders are available with DDLab (section 3.6.1) for some combinations of rule type,  $v$ ,  $k$ , and lattice dimensions<sup>5</sup>. A rule from such a numbered collection can be loaded on-the-fly (key **g**) while space-time patterns are running (section 32.6.1). If a collection is detected for the current  $v$ ,  $k$ , the following top-right prompt is presented for either a random or consecutive number from the list,

**v3k6 complex rules (select on-the-fly with “g”)**  
**consecutive enter start index 1-54, rnd-def: (for example)**

---

<sup>5</sup>Although the complex rules relate to specific dimensions, they often result in interesting dynamics in other than the intended dimensions.

Enter **return** for a rule picked randomly from the list, or a number to start consecutive selections, to be loaded later on-the-fly. Consecutive list numbers will cycle back to one on reaching the maximum number. In the example above there are 55 rules in TFO-mode, for kcode *v3k6*, intended for 2d. This option is also available from the interrupt/pause prompt (key **G**, section 32.16.1).

### 31.2.10 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 highest *v*-ary value of value-frequencies in the neighborhood is on the left (section 13.6.1) and the lowest on the right. To allow kcode with the opposite convention to run as intended, the following top-right prompt is presented,

**invert kcode value order -i:**

Enter **i** to run the kcode according to the inverted convention, or **return** for the standard convention. The kcode itself is not changed, just the way it is accessed while running forward. This is analogous to inverting the rcode in section 18.3 but in that case the rcode itself is changed.

## 31.3 Periodic or Null boundary conditions

Enter **b** at the first output parameter prompt (section 31.1) to skip directly to the “boundaries” option for setting null boundary conditions (NBC) instead of the default periodic boundary conditions (PBC). Space-time patterns with either PBC and NBC apply to 1d, 2d and 3d networks. The difference between PBC and NBC is described in sections 2.7 and 26.1.

The following top-right prompt is presented,

**boundaries: Periodic boundaries, set Null-N:**

Enter **N** for null boundary conditions. Note that once running space-time patterns, PBC and NBC can be toggle on-the-fly with key | (pipe or vertical bar).

## 31.4 Asynchronous and noisy updating

By default, updating the next time-step is deterministic and synchronous, but can be made noisy and asynchronous.

Enter **u** at the first output parameter prompt (section 31.1) to skip directly to the “update” category of options for probabilistic, sequential and partial order methods of updating, or arrive there by viewing the output parameters in sequence. The following top-right prompt is presented,

**updating: sequential-s, partial-order-p, probabilistic-def:**

Enter **return** to move directly to the prompt for probabilistic updating (section 31.4.1) which is also reached after sequential or partial-order prompts, or enter **s** or **p** for the prompts to set various types of sequential or partial-order updating (sections 31.4.2 and 31.4.3). There can be any combination of these settings, and they can also be toggled on-the-fly (section 32.4).

### 31.4.1 Probabilistic updating

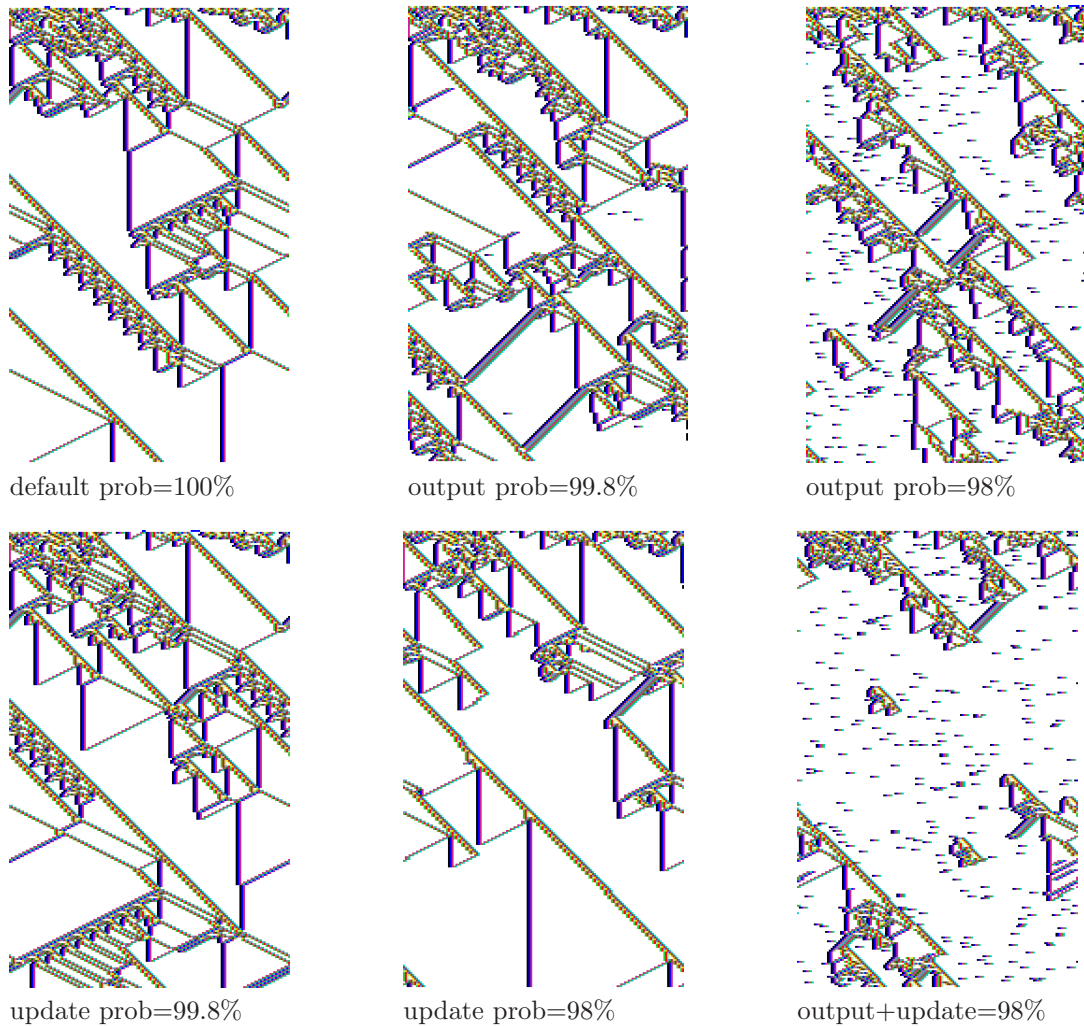


Figure 31.6: Probabilistic updating: the “update” probability that a cell will be updated at all, and the “output” probability that updating obeys the rule-table, otherwise update randomly (section 31.4.1). For  $v2k5$  CA rcode(hex)e9f6a815,  $n=150$ , 245 time-steps from the same initial state. Cells colors are by neighborhood (section 31.2.1) and filtered (section 32.11) to emphasize gliders.

There are two types of probabilistic or noisy updating methods, where the probability, by default 100%, can be reset to a lower value,

update probability ... where each cell updates with a given probability at each time-step. This is very similar to partial order updating (section 31.4.3). If this probability is set to  $P$ , there is a  $1-P$  chance that a cell is not updated but stays the same.

output probability ... where each cell updates correctly with a given probability at each time-step. If the probability is set to  $P$ , there is a  $1-P$  chance that a cell's value will be randomly assigned instead of respecting the dynamical rule, which introduces noise.

Enter **return** at the updating prompt (section 31.4) for the following top-right prompt, or arrive there by viewing the output parameters in sequence,

```
update prob (def 95.00%) 0-100:
output prob update (def 95.00%) 0-100:
```

First set 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 will be subject to the output probability. Figure 31.6 gives examples. Both probabilities can be toggled on-the-fly (section 32.4). If a probability is set it will be active when space time-patterns start.

### 31.4.2 sequential updating — space-time patterns

In sequential updating, each cell is updated in turn in some arbitrary order — then the “state” is the configuration when all  $n$  updates are complete. There are  $n!$  possible sequential updating orders in a network of size  $n$ . As for attractor basins (section 29.9), DDLab provides simple default orders, forward or backward along the network index, a random order (figure 31.7a), and a specific order set by hand. A random order can also be set to change at each iteration. The order can be saved and loaded from a `.ord` file. All possible orders can be listed (as in section 29.9.3) but this applies for  $n \leq 12$  only.

These options are essentially the same as for setting sequential updating for attractor basins, except that the limits on  $n$  in table 29.1 do not apply. Synchronous/sequential updating (whatever order was set) can be toggled on-the-fly (section 32.4) as in figure 31.7. If sequential updating was not selected, toggling will give the default left to right order.

Enter **s** at the updating prompt (section 31.4) for the following top-right sequential updating prompt<sup>6</sup>,

```
sequential update: >-(def) <-b rnd-r rernd-R set-s all-S: (all-S if n ≤ 12)
```

These options are summarized below — the detailed descriptions are to be found in section 29.9.

<i>options</i> ...	<i>what they mean</i>
<b>&gt;-(def)</b> ...	enter <b>return</b> 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 10.2).
<b>&lt;-b</b> ...	for the opposite order, from right to left, cell index 0 to $n-1$ .
<b>rnd-r</b> ...	for a random order presented in a top-right window including saving and loading the order (section 29.9.1).
<b>rernd-R</b> ...	for a new random order at each iteration.
<b>set-s</b> ...	to set a specific order by hand, including saving and loading the order (section 29.9.2). $n \leq 630$ normally fits into the “by hand” window.
<b>all-S</b> ...	(if $n \leq 12$ ) to list all $n!$ possible orders and select a specific order from the list (section 29.9.3).

---

<sup>6</sup>The sequential updating prompt for space-time patterns is similar to the sequential updating prompt for attractor basins (section 29.9).

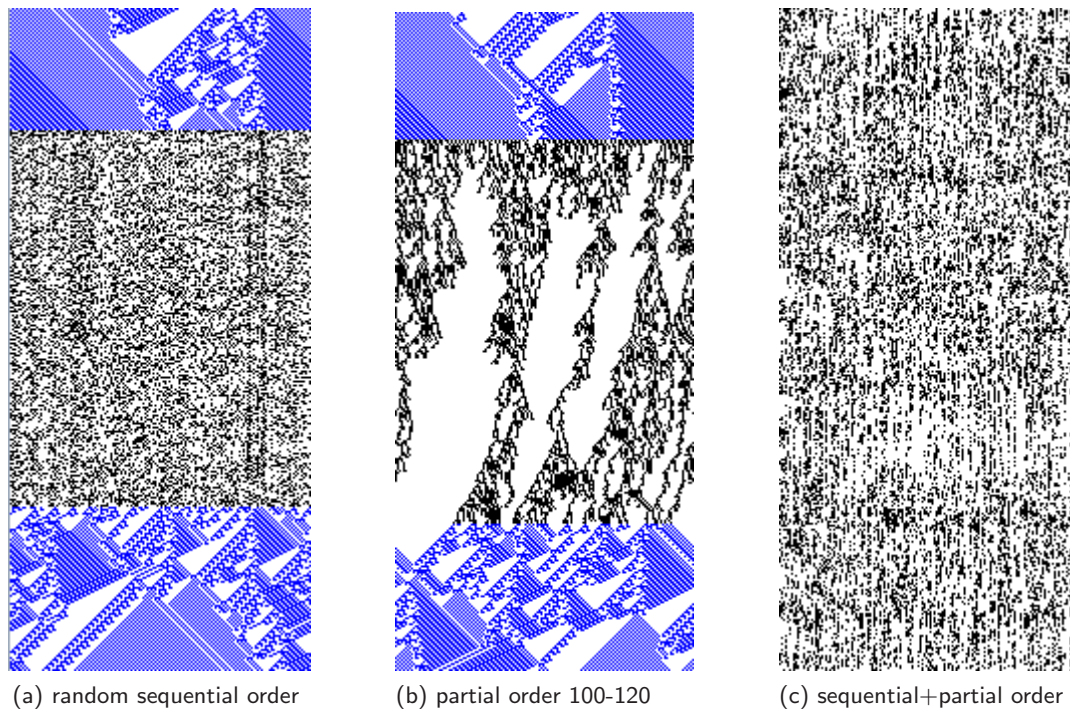


Figure 31.7: Examples of updating with sequential, partial order, and both orders combined, for a 1d CA  $v2k5$  `rcode(hex)bc82271c`,  $n=150$ . In case (a) and (b) parallel updating was switched to the new order on-the-fly, for about 180 time-steps, then back to parallel. In case (c) about 310 time-steps are shown with combined sequential and partial order updating

### 31.4.3 partial order updating — space-time patterns

Partial order updating is when a subset of cells update in parallel, followed by the next subset — then the “state” is the configuration after each updated subset. Lower and upper limits define an “update subset” between 1 and network size  $n$  — the default is  $n$ . At each time-step, a random size is set between these limits, and applied randomly to positions in the network — only these are updated. This is very similar to “update probability” (section 31.4.1) but the partial order method guarantees a fixed range of the update subset, including just one constant value if lower and upper limits are the same ( $\text{min}=\text{max}$ ).

Enter **p** in section 31.4 to set partial order updating. The following top-right prompt for the minimum and maximum size limits of the update subset are presented in turn (for  $n=150$ ),

**partial order updating: update subset limits (def 1:150): min: 100 max:120**

In this example (figure 31.7b), an update subset  $s$  of between 100 and 120 ( $s$  set randomly) will be assigned at random to the network. Only these cells are updated to make the next state, and the process is repeated at each iteration.

Synchronous/partial order updating can be toggled on-the-fly (section 32.4) — if partial order updating was not defined, toggling gives the default update subset. Sequential updating



(section 31.4.2) can be active at the same time as partial order updating (figure 31.7c) — in this case the partial order subset will be updated sequentially at each iteration to make the next state.

---

## 31.5 Input-entropy and pattern density

Measures on space-time patterns, taken over a window of time-steps and shown graphically, provide some useful insights into dynamics and rule-space [33, 44, 38]. The measures are as follows,

- input-frequency ... the frequency of rule-table lookups, or neighborhood size blocks — provides the basis for entropy measures and filtering space-time patterns. To increase block size, a rule can be transformed to an equivalent rule with a larger neighborhood (section 18.7.1).
- input-entropy ... the Shannon entropy of the input-frequency distribution — provides the basis for classifying rule-space (chapter 33).
- pattern density ... the fractions of different values — provides an entropy-density scatter plot (figure 32.32) — a characteristic signature of each complex CA.

The measures apply either to the whole network or to one selected cell. Although input-entropy is intended for a homogeneous-rule network, it works for a rulemix provided  $k$  is homogeneous. For a  $k$ -mix a single position in the network must be selected (for both entropy and density) — but this can be done in all cases if required.

Note that the entropy and density both become active if either is selected, but interactive plots of the selected measure will appear first, alongside space-time patterns — thereafter the two can be toggled on-the-fly (section 32.12.3).

Enter **e** in section 31.1 to skip directly to the “entropy” category of options, or arrive there by viewing the output parameters in sequence. The following top-right prompt is presented,

```
for native 1d networks with homogeneous-k
analysis: none-n, pattern-density -p
input-frequency/entropy-(def):

for native 2d or 3d networks, or mixed-k
analysis: none-(def) pattern-density -p
input-frequency/entropy-e:
```

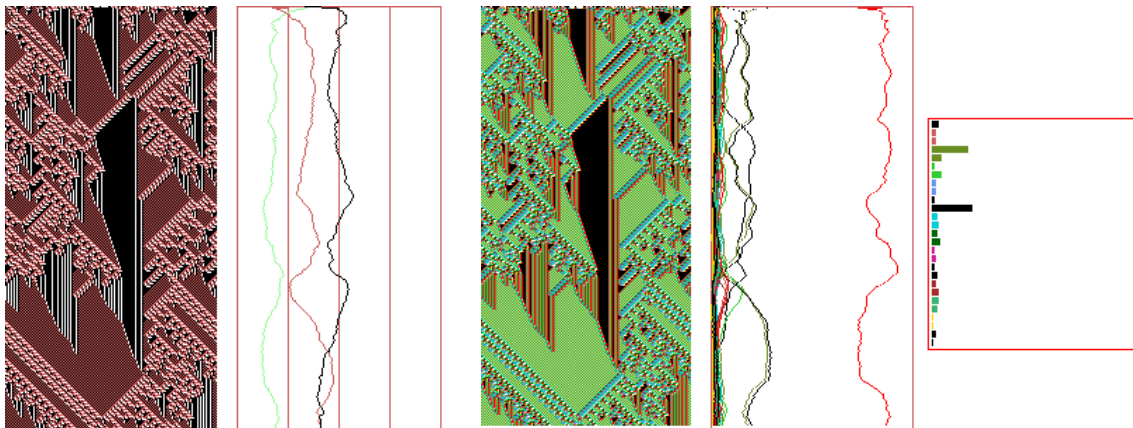
Input-frequency and pattern-density can be toggled on-the-fly, as well as other aspects of the presentation (section 31.5.3).

### 31.5.1 Single cell input-entropy and pattern density

If either input-entropy or pattern density was set, for homogeneous- $k$  the next prompt allows a single cell to be selected,

```
single-s all-(def): (for homogeneous-k )
```

If a single cell was selected above, or for a mixed- $k$  network, the next prompt selects the position of the cell in the network,



Pattern density plots of each value, alongside space-time patterns in value colors

Input-Entropy plot and input-frequency histogram (and plots), alongside space-time patterns in neighborhood colors.

Figure 31.8: The pattern density and input-entropy plots, for the same *v3k3* 1d CA, rcode (hex)2698a14246a466,  $n=150$ , 300 time-steps from the same random initial state. Space is across and time down as in figure 2.4. Measures were averaged over a moving window of 10 time-steps.

*Left:* the pattern density alongside space-time patterns in value colors — the density of each value is plotted on the x-axis which has 4 divisions. The density plots correspond to value colors except zero which is colored light green.

*Right:* the input-entropy (in red) alongside space-time patterns shown in neighborhood colors — the entropy is on the x-axis, with zero entropy on the left and maximum entropy on the right. The input frequency histogram with 27 bars is on the extreme right showing the frequency of each neighborhood, with the zero neighborhood at the bottom. In this example the input frequencies are also plotted on the input-entropy graph, toggled on-the-fly (section 32.12.4).

*for a 1d network  $n=150$*

**single:** enter index (max 149 def 0):

*for a 2d network  $40 \times 0$*

**single 2d:** 2d index (max 39,39, now 0,0)

enter i:      enter j:

*for a 3d network  $22 \times 22 \times 22$*

**single 3d:** 3d index (max 21,21,21, now 0,0,0)

enter i:      enter j:      enter h:

This can be amended on-the-fly (section 31.5.3).

### 31.5.2 Generation size - moving window of time-steps

The next prompt sets the analysis generation size, the size of the moving window of trailing time-steps over which the measures are averaged. For input-entropy these are past time-steps, for pattern-density the present time-step is also included. The initial default is 10 for the whole network, and 500 for a single cell — the new setting becomes the default,



**enter new 'analysis' generation size (now 10):** *(for the whole network)*

This can be amended on-the-fly (section 31.5.3).

### 31.5.3 On-the-fly changes to input-entropy and pattern density

Once space-time patterns are running, and either input-entropy and pattern density was activated, on-the-fly options (section 32.3) apply to make changes as follows,

*key ... on-the-fly option*

**s** ... toggle between input-entropy and pattern density.

**G** ... change the current analysis generation size (section 32.12.6). which is displayed in the on-the-fly key index (sections 32.1 and 32.3), for example, **G..a-gens=10**.

If a single cell was set above, its position is also shown and can be changed, for example,

**G..a-gens=10 pos=9** ... for 1d

**G..a-gens=10 pos=5,6** ... for 2d

**G..a-gens=10 pos=8,9,10** ... for 3d

*if input-entropy is displayed*

**0** ... toggle input-frequency histogram bars between normal width and 1 pixel — useful for large rule-tables.

**%** ... show the input-frequency histogram with a time dimension — if the number of bars  $\leq 64$  (figure 32.31 and section 32.12.1).

**)/(** ... amplify the input-frequency histogram, and restore default.

**j** ... 3-way toggle between input-entropy, input-frequencies as a set of graphs, and both together (section 32.12.4).

**u** ... tog entropy-density scatter plot, where rules have characteristic signatures (figure 32.32 and section 32.12.5).

**E** ... *(while the entropy-density scatter plot is active)* a plot of mean entropy against entropy variability<sup>7</sup> for a sample of rules, giving a 3d histogram that separates out order, complexity and chaos (chapter 33).

---

## 31.6 Damage, the difference between two networks

*1d and 2d networks only — not 3d*

It's 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/value 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/value. This option applies to 1d and 2d networks only.

---

<sup>7</sup>Entropy variability is either a min-max measure — the maximum entropy up-slope found, or standard deviation.

Damaged cells can be defined in two ways. Once damaged, keep the damage irrespective of future dynamics, or simply the instantaneous difference at each time-step — this can be toggled on-the-fly (key “=” section 32.12.10).

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/value, shown as a histogram. This is one method of characterizing order and chaos in network dynamics, especially in random Boolean networks applied as idealized models of genetic regulatory networks [16, 36].

Enter **m** at the first output parameter prompt (section 31.1) to skip directly to the “damage” category of options, or arrive there by viewing the output parameters in sequence. The following top-right prompt is presented,

**damage: show difference between 2 networks**  
**keep damage-f, don't keep-F, histograms: actual-a bins-b:**

Enter **a** or **b** to shown a histogram of damage spread for a sample of initial states generated automatically (section 31.6.2) — **a** gives the actual damage sizes, **b** gives damage falling into a number of preset bins. Keeping the damage applies — once damaged, a cell stays damaged.

Otherwise, for a manual version of damage spread (without a sample or histogram) enter **f** to keep the damage, or **F** for the instantaneous difference at each time-step.

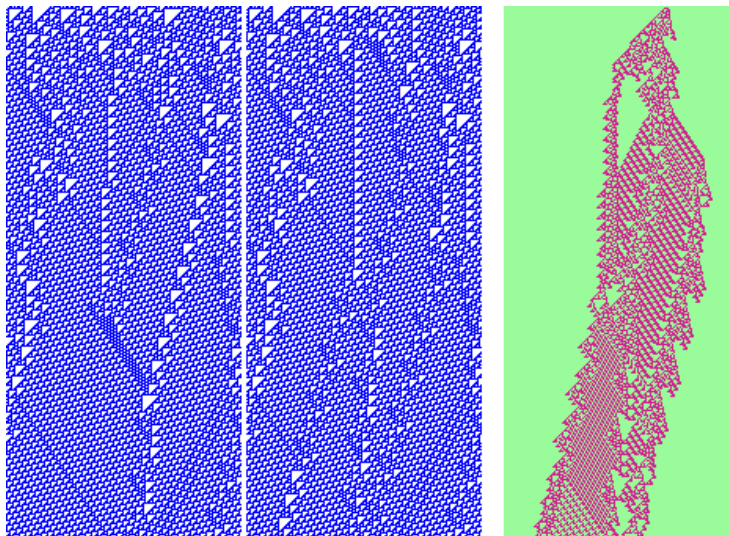


Figure 31.9: The difference in the dynamics between two 1d CA. *Left*: the space-time patterns of two identical CA (color by value), except that the random initial state differs by one bit. *Right*: the instantaneous difference between the two at each time-step shown as a third space-time pattern. *v2k3* rcode (dec)110,  $n=150+150$ .

### 31.6.1 Duplicate the network and seed

A subsequent option, normally selected together with the “damage” option (section 31.6) 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 31.6) and duplicating the network can be used independently<sup>8</sup>. If “damage” was selected and the network is not duplicated, the original network and its initial state will simply be notionally 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,

**can't divide odd width network, cont-ret:**

Once the network is duplicated, prompts are presented in turn to look at and possibly modify the duplicated wiring (section 17.1), and the duplicated seed ((section 21.1). Figures 31.6.1 and 21.1 give examples. 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 31.6.1.

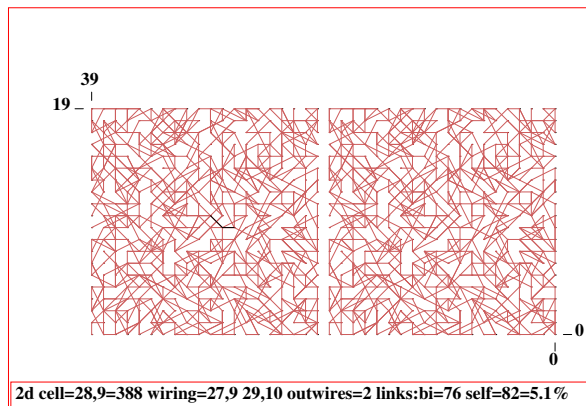


Figure 31.10: A duplicated 2d RBN,  $40 \times 20$ ,  $v2k2$ , showing just links, made up of two identical  $20 \times 20$  sub-networks. The original  $20 \times 20$  network was set with random wiring confined to a 5 cell local zone (section 12.5.2) and with periodic boundaries suppressed (section 12.5.5).

The display shown here is achieved by first defining a block equal to the whole network (enter **a**, section 17.7.5), then applying a 5-way toggle to show links only (enter **n**, section 17.7.3 and figure 17.11(5)).

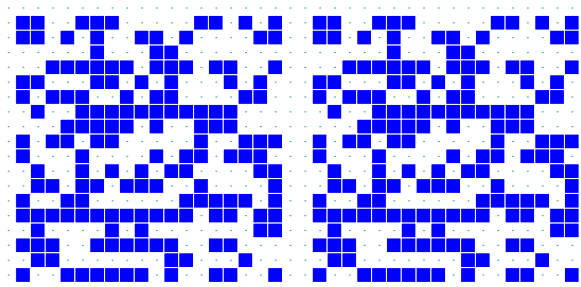


Figure 31.11: A duplicated 2d seed,  $40 \times 20$  consisting of two identical  $20 \times 20$  sub-seeds

<sup>8</sup>This can be repeated to re-double the network making 4, 8, etc independent networks, but the total cell count cannot exceed  $n_{Lim}=65025$  (section 8.3)

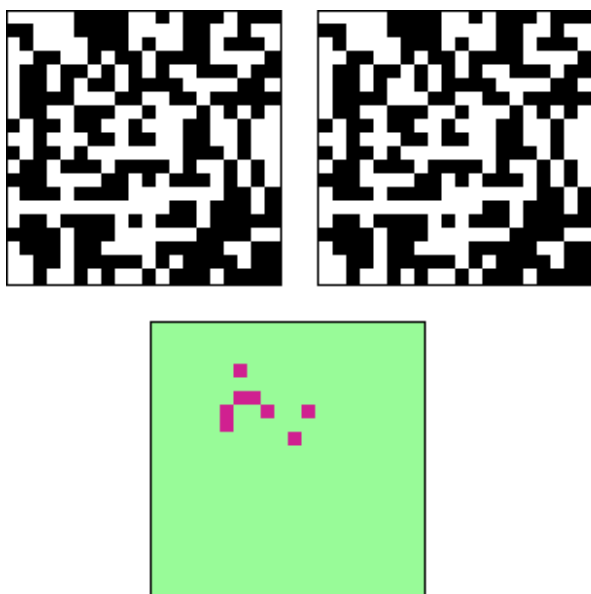


Figure 31.12: An example of 2d “damage”. The difference in dynamics between two 2d RBN. *Above*: the space-time patterns of two identical  $20 \times 20$  RBN (color by value), except that one bit was flipped in the initial state. The RBN has  $k=2$ , with connections as in in figure 31.6.1. *Below*: 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 31.6, so that once a cell is “damaged” - it stays “damaged”.

Note that if `f` or `F`, for a manual version of damage spread, was selected in section 31.6, and the network was duplicated, then there will be no difference in the initial states, thus no damage. To introduce a difference, a random bit/value flip can be made on-the-fly (key `l`, section 32.8.2). A new random initial state (the same for each sub-network) can be set on-the-fly (key `4`, section 32.8.1).

## 31.6.2 The Damage Histogram

*1d and 2d networks only — not 3d*

Enter `a` for actual, or `b` for bins, in section 31.6, to generate a histogram of damage spread for a sample of initial states generated automatically. The prompt to duplicate the network (section 31.6.1) is presented. Do not duplicate if this was done previously, otherwise the duplicated network will be reduplicated, making 4 copies of the original.

Top-right prompts are presented in sequence as follows to set the histogram parameters. The second line differs between actual or bins. The defaults may vary and some can also be reset later during a histogram pause (section 31.6.4). This example is for two  $20 \times 20$  sub-networks.

### Damage Spread Histogram

**cut-off size (min 20, def 400):** *(for actual damage, a entered in section 31.6)*

**delay damage, time-steps before damage start (def 0):**

**define damage, time-steps same damage (def 5):**

*(for damage bins, the second line is as follows, b entered in section 31.6)*

**no of bins (min 10, def 100):**

These options are described below,

options ... what they mean

- cut-off size ...** (for actual damage only) set the cut-off or accept the default, where any damage above the cut-off is lumped together in one excess bin.
- no of bins ...** (for damage bins only) set the number of bins or accept the default.
- delay damage ...** set the number of time-steps before the two sub-networks are compared and the measure of the damage starts — the default is 0.
- define damage ...** to define 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 reset.

### 31.6.3 Drawing the damage histogram

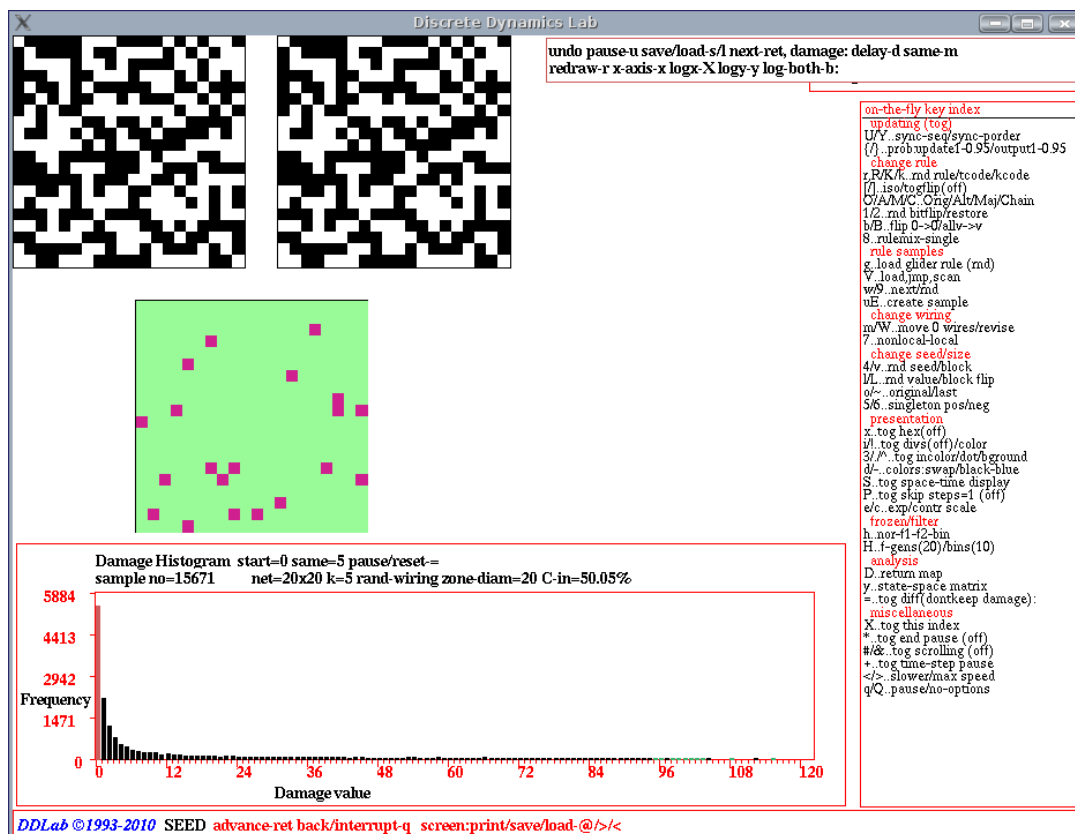


Figure 31.13: The DDLab screen, showing the automatic process for gathering statistics on (actual) damage. This example is for two  $20 \times 20$  RBN,  $k=5$ , with canalizing inputs set to 50%, (chapter 15). At each pass, a new random initial state is assigned to the sub-networks, differing by one 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. The process was paused with the = key.

The histogram is drawn in a window in the lower part of the screen, as in figure 31.13 and 31.14. At each update, a new unbiased random initial state is assigned to the sub-networks, differing by one bit/value at a random position. When the damage stops spreading (see “define damage” in section 31.6.2), the histogram is updated — plotting the damage size ( $x$  axis) against the frequency of each size ( $y$  axis). 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  $\overline{\text{damage}}=11/400=2.8\%$ . To considerably speed up computation for generating these histograms, the display of space-time pattern graphics can be toggled off on-the-fly (key **S**, section 32.9).

Information and reminders shown within the histogram window, are described below,

*info/reminders ... what they mean (values shown are examples)*

**start=0** ... number of time-steps before the measure of damage begins.

**same=5** ... number of time-steps defining damage as having stabilized.

**pause/reset=** ... enter = (the equals sign) to pause and reset the histogram parameters.

**sample no=15671** ... the size of the sample so far.

**net=20×20** ... network size.

**k=5** ... neighborhood size  $k$ , or “**k=3 to 6**” for mixed- $k$ .

**local wiring** ... local wiring, as in CA.

**rand-wiring** ... random wiring, as in RBN.

**zone diam=9** ... the size of the local zone confining random wiring (section 12.5.2).

**C-in=50.00%** ... the percentage of canalizing inputs.

Enter = (the equals sign) to pause the histogram and pause options (section 31.6.4). Alternatively, pause by entering **q** for the general pause options for space-time patterns (section 32.14). Most on-the-fly options (chapter 32) can be activated during the histogram routine. To speed the damage histogram, toggle the space-time graphics off with the on-the-fly option **S** (section 32.9.8), which also helps to avoid a graphics clash in 1d.

### 31.6.4 Pausing the damage histogram

While the histogram is being drawn (section 31.6.3), enter = (the equals sign) to pause, the following top-right prompt is presented to amend parameters and presentation,

**stp-q undo-pause-u save/load-s/l next-ret, damage: delay-d same-m**  
**redraw-r xcut-off-X logx-x logy-y log-both-b:** (*xcut-off-x for actual damage only*)

These options are explained below,

*options ... what they mean*

**stp-q** ... for the space-time pause prompt with many options (section 32.14). Enter **q** again to backtrack, or **return** to the histogram pause prompt.

**undo pause-u** ... to continue the histogram without pause.

**save/load-l/s** ... enter **s** to save the histogram data as a **.his** file (section 35.3) — the file holds the current frequencies (0 to max  $x$ -axis) in successive pairs of bytes. Enter **l** to load and print the data in the terminal (not for DOS).

**next-ret** ... enter **return** to continue the histogram, but pausing after each sample.

**delay-d** ... to alter the damage delay (section 31.6.2).

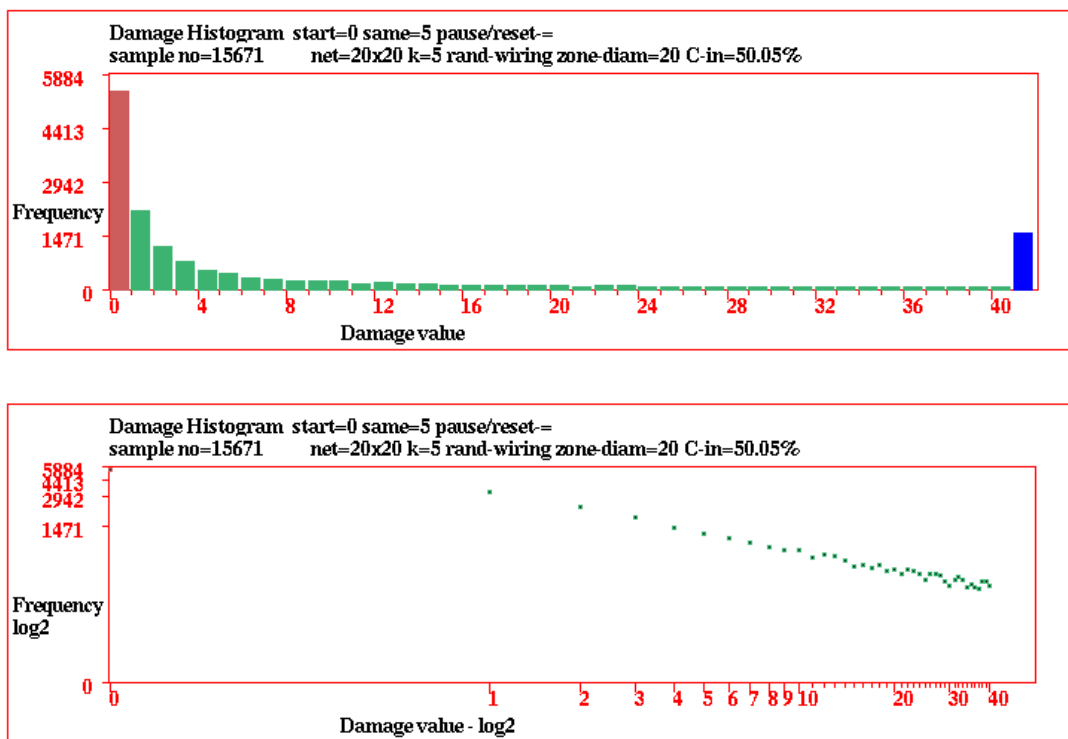


Figure 31.14: The damage histogram showing actual damage, for the same network as in figure 31.13. *Top*: with the damage cut-off set at 40, the frequency of damage of 41 and above is represented by the blue bar on the extreme right. *Bottom*: the damage histogram as a log-log plot.

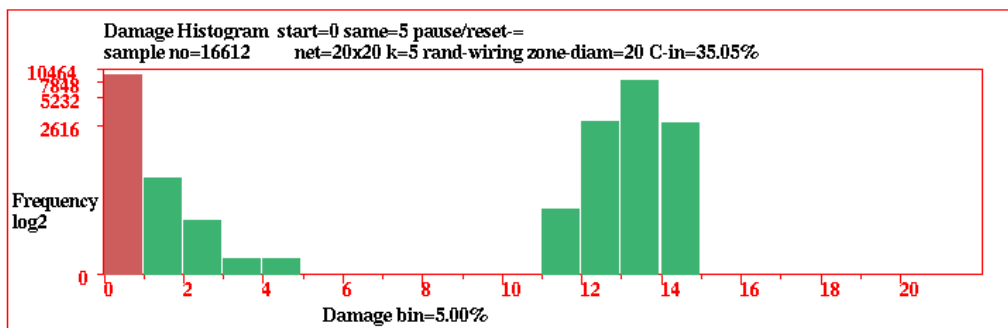


Figure 31.15: The damage histogram with the damage frequency allocated to 20 equal size bins, shown with the  $y$  axis in log form. This is a similar network as in figure 31.13, but with canalyzing inputs set to 35%, resulting in less ordered dynamics and a bimodal distribution of damage frequency.

- same-m** ... to redefine damage, the number of time-steps defining damage as having stabilized (section 31.6.2).
- redraw-r** ... to redraw the histogram, which may have been overwritten by other graphics.
- xcut-off-X** ... (*for actual damage only*) to re-scale the x-axis, the following top-right prompt is presented,
- revise x-cut-off, now 400 (min 20, def 400):** (*for example*)
- enter the new cut-off. Any damage above the cut-off is lumped together in one excess bin.
- logx-x, logx-y** ... to show the  $x$ -axis or  $y$ -axis separately in log form, with the following top-right reminder,
- x-axis log2, cont-ret:** or **y axis log2, cont-ret:**
- log-both-b** ... to show both the  $x$  and  $y$ -axis in log form, a log-log plot (as in figure 31.14 *Bottom*), with the following top-right reminder,
- xy axes log2, cont-ret:**

## 31.7 Attractor histogram

A basin of attraction field cannot be computed for networks larger than the limits in section 7.3. Even within these limits, larger sizes may not be practical because of probable time/memory constraints. However, statistical data on the range of attractors and their relative sizes for networks well beyond the limits in table 7.3 can be obtained by automatically running a network forwards from many random initial states, identifying different attractors and creating a histogram of the frequency of each type. The largest basins will be found the most frequently — small basins may be missed. Attractors are identified by comparing and recording attractor states, and the data gathered includes the period and the average transient length — the number of time-steps to reach the attractor. A new feature analyses the transient data to record the set of unique transients states to each attractor, without duplication (figure 31.19), giving a closer approximation to the actual basin of attraction field than previous attractor histogram data.

The attractor histogram is suitable for ordered networks — that have short transients and attractor cycles where the attractor can actually be reached and disclosed in a reasonable time, so is not practical if the dynamics are too chaotic — in that case the “skeleton” histogram is potentially an alternative method (section 31.8).

The method has been applied in RBN models of genetic regulatory networks [16, 35] with a critical degree of canalizing functions (chapter 15) as disclosed by the Derrida plot (chapter 22). For CA, the method is also useful for ordered and complex (glider) rules<sup>9</sup> with very high in-degree — which is a limitation on reverse algorithm methods for generating attractor basins (section 2.19).

Enter **a** at the first output parameter prompt (section 31.1) to skip directly to the “attractor” category of options (the only way to arrive there) to generate the attractor histogram. The following top-right reminder prompt is presented,

**attractors: histogram, include trans data-t, cancel-q cont-ret:**

<sup>9</sup>For example, ordered rules such as majority rules (section 16.7) or Altenberg rules (section 16.9), complex (glider) rules (section 3.6.1), or the game-of-Life (sections 16.10 and 14.2.2).



Enter **return** — for  $v \geq 3$  the histogram will commence, for  $v=2$  there is a further option (section 31.7.1, below).

### 31.7.1 Density classification problem — attractor histogram *for binary systems ( $v=2$ ) only*

For binary systems ( $v=2$ ) only, there is a further top-right prompt for the inclusion of data on initial state densities that fall into the all 0s or all 1s point attractors,

**include seed density data for all=0/1s point attractors-a:**

Enter **a** to include this data, which is intended mainly to check the fitness of rules that have been evolved to solve the “density classification problem”, a favourite exercise in emergent computation [12, 18], where initial states with densities  $d < 0.5$  are evolved to fall into the all 0s point attractor, and  $d > 0.5$  to the all 1s point attractor ( $d = 0.5$  for even  $n$  are also included in DDLab). An alternative method for density classification based on attractor basins is described in section 24.7.

### 31.7.2 Drawing the attractor histogram

The histogram is drawn in a lower-center window. Space-time patterns (in 1d or 2d) appear in the top-left as usual, from each new initial state. The histogram is continuously updated (in black), 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. The axes rescale automatically (and columns turn green) on reaching the current scale limits.

At the same time, a top-right data window gives current data: the current total of attractors found, the current type, its period and average transient length, for example,

**types=44 this=8 attperiod=12 avtrans=14.0**

Most space-time pattern on-the-fly options work during the histogram routine. To speed the attractor histogram, toggle space-time graphics off with on-the-fly option **S** (section 32.9.8).

#### 31.7.2.1 Histogram window information

Information is also displayed at the top of the histogram window (figure 31.18), including some reminders about the network, decoded as follows (values shown are examples),

*info* ... *decode*  
**pause/reset=** ... reminder to enter = (the equals sign) to pause, reset, sort, show data, and other options (section 31.7.3).  
**sample no=3914** ... the size of the sample so far.  
**net=20×20** ... network size for 2d, or **net=200** for 1d.  
**k=5** ... neighborhood size,  $k$ , or **k=3-5** for a mixed- $k$ .  
**local wiring** ... for local wiring, as in CA.  
**rand-wiring** ... for random wiring, as in RBN.  
**zone diam=20** ... the size of the relative local zone within which random wiring is confined, see section 12.5.2.  
**C-in=50.00%** ... the percentage of canalizing inputs (section 15).

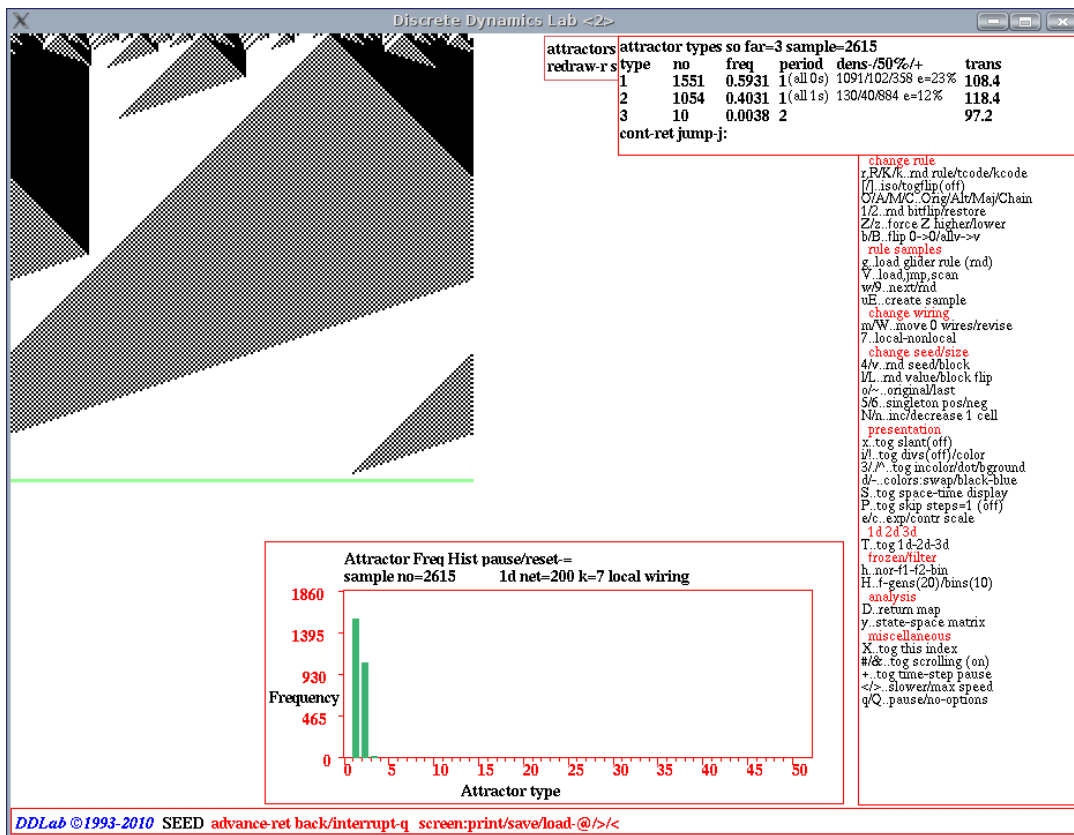


Figure 31.16: A snapshot of the DDLab screen, showing the attractor histogram for density classification, applying the Raja Das rule (as in figure 24.7) to a 1d CA  $v2k7$ ,  $n=200$ . *Top left*: a sample space-time pattern (scale enlarged) leading to the all-0s point attractor (repeated in green). *Top Right*: a window showing data on the three attractors found, giving the fraction densities ( $d < 0.5$ ,  $d = 0.5$ ,  $d > 0.5$ ) arriving at the all 0s and all 1s point attractors — the data is described in section 31.7.5.3. *Bottom center*: the attractor histogram for a sample of 2615. As  $n$  is even, a third (infrequent) 2-state attractor, consisting of alternating values 0101... and 1010..., is able to exist.

### 31.7.3 Pausing the attractor histogram

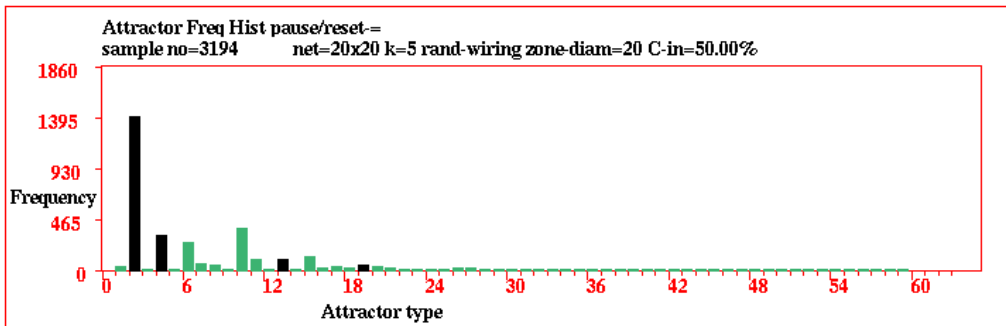
The space-time pattern on-the-fly pause (key **q**, section 32.14) applies while drawing the histogram.

However, there is a special pause for the histogram itself, `enter =` (the equals sign) for the following top-right prompt which provides a range of histogram options,

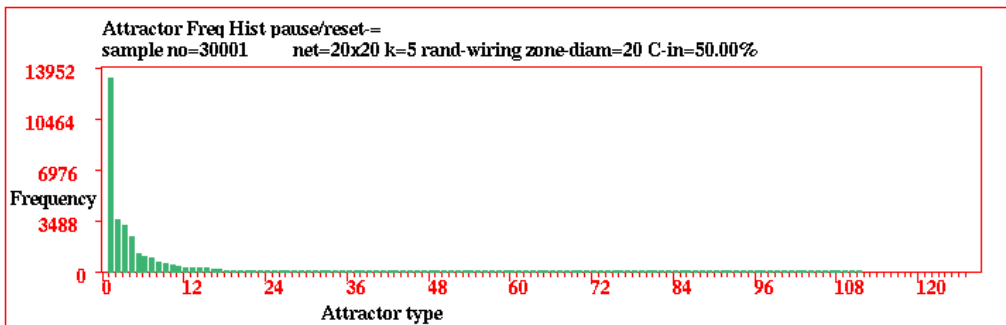
**attractors histogram: hist-h stp-q, data:show-d print-p save-S  
 redraw-r sort-s data-d undo-pause-u next-ret jump-graph-j:**

These options are summarized below, some described further in the sections indicated<sup>10</sup>,

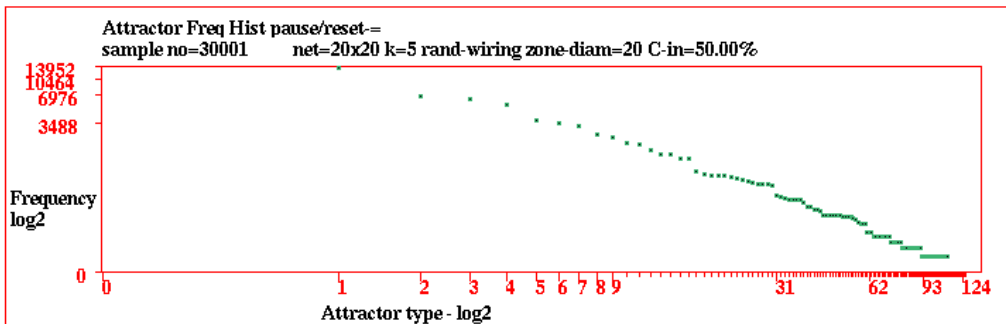
<sup>10</sup>These option are the same or similar to the skeletons histogram options in section 31.8.3.



unsorted histogram, for a sample of 3194

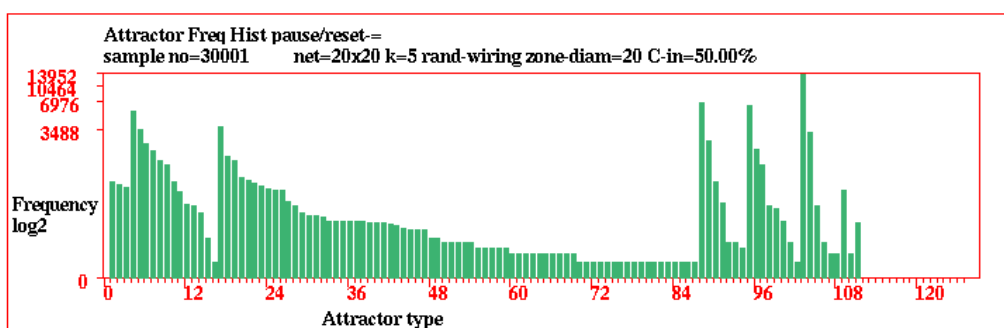


sorted histogram, according to frequency, for a sample of 30001, with 111 attractors

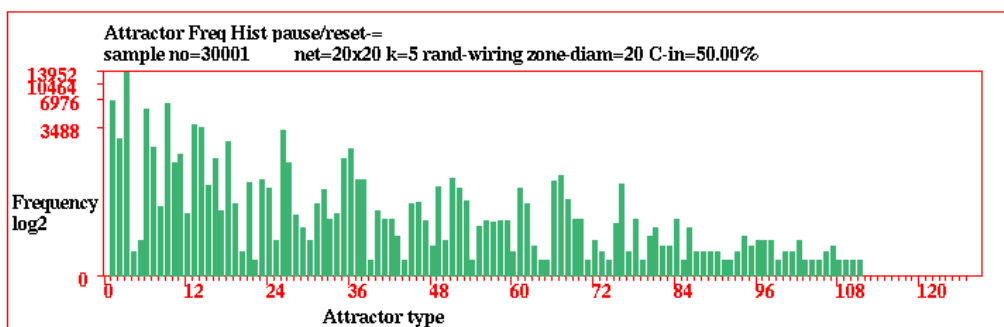


The sorted histogram, shown as a log-log plot

Figure 31.17: The attractor frequency histogram, for a  $20 \times 20$  RBN,  $v2k5$ , with unbiased random wiring, and analyzing inputs set to 50% (chapter 15). Each sample starts with a new unbiased random initial state, and the attractor type into which the transient falls is identified, either as an existing type, or a new type which is added to the list of existing types. The histogram columns are continuously updated, in black, showing the attractor type ( $x$ -axis) against the frequency of arriving at that type ( $y$ -axis). Given a large enough sample, bar height indicates the relative size of the basin of attraction. The axes rescale automatically on reaching the current scale limits. Columns turn green when rescaled, redrawn, or sorted.



first sorted by frequency (figure 31.17), center) then by period



first sorted by frequency (figure 31.17), center) then by average transient

Figure 31.18: The attractor histogram for a sample of 30001 and 111 attractors, sorted by attractor period and average transient length, with log  $y$ -axis. (section 31.7.4).

*options ... what they mean*

**hist-h** ... further histogram options: to save/load, rescale, and log-plot (section 31.7.4).

**stp-q** ... for the space-time pause prompt with many options (section 32.14. Enter **q** again to backtrack, or **return** to the histogram pause prompt.

**data:** ... attractor data — for 1d networks print/save includes listing attractor states.

**show-d** ... show attractor data in the DDLab screen (section 31.7.5).

**print-p** ... print the attractor data in the terminal (section 31.7.6).

**save-S** ... save the attractor states as a .dat file (section 31.7.6).

**redraw-r** ... to redraw the histogram, which may have been overwritten by other graphics.

**sort-s** ... sort the histogram by frequency and other measures (section 31.7.7).

**undo-pause-u** ... to continue the histogram without pausing.

**next-ret** ... enter **return** to continue the histogram, but pausing after each sample.

**jump-graph-j** ... to construct a jump-graph from the histogram data (section 31.7.8).

### 31.7.4 Rescaling the attractor histogram

If **h** for further histogram options is entered in section 31.8.3, or after sorting (enter **s**), the following top right prompts are presented,

```
save/load-l/s undo-pause-u cont-ret
x-axis-x logx-X logy-y log-both-b:
```

These options are explained below,

- | <u>options</u> ...        | <u>what they mean</u>  |
|---------------------------|--|
| <b>save/load-l/s</b> ...  | enter <b>s</b> to save the histogram data as a <b>.his</b> file (section 35.3). Enter <b>l</b> to load the data and see it in the terminal (not for DOS). The file holds frequencies (0-max) in successive pairs of bytes. |
| <b>undo-pause-u</b> ...   | continue the histogram without pause.  |
| <b>cont-ret</b> ...       | <b>return</b> to the histogram pause prompt (section 31.7.3).  |
| <b>x-axis-X</b> ...       | rescale the x-axis — zoom into the left of the histogram. The following top-right prompt is presented <i>for example</i> ,   |
|                           | <b>revise x-cut-off, now 63 (min 20, def 63):</b>  |
| <b>logx-x, logx-y</b> ... | to show the <i>x</i> -axis or <i>y</i> -axis separately in log form, with the following top-right reminder,  |
|                           | <b>x-axis log2, cont-ret:</b> or <b>y axis log2, cont-ret:</b>   |
| <b>log-both-b</b> ...     | show both the <i>x</i> and <i>y</i> -axis in log form (as in figure 31.17, bottom), with the following top-right reminder,   |
|                           | <b>xy axis log2, cont-ret:</b>   |

### 31.7.5 Attractor histogram data

Enter **d**, **p** or **s** in section 31.7.3 to show the data in the DDLab screen (section 31.7.5.1), or print to the terminal (DOS only) or save to file 31.7.6. The data gives a summary for each attractor according to the histogram order, so sorting the histogram beforehand is advisable. For 1d networks, print/save data also includes a list of the states in each attractor.

#### 31.7.5.1 Attractor histogram screen data

Enter **d** in section 31.7.3 to show a list of attractor types, as many as will fit, displayed in a window on the right of the screen, followed by further options. The list order depends on whether the list was sorted in section 31.7.7, if not, the list is ranked in the order that attractor types were found.

The following example is for the sorted histogram in figure 31.17,

```

attractor types so far=111 sample=30001
type  no    freq    period  trans
1     13041  0.4347  4       54.1
2     3610   0.1203  8       77.0
3     3095   0.1032  6       40.0
... (the list continues)
39    22     0.0007  12      33.3
40    21     0.0007  12      24.8
41    20     0.0007  16      27.4
more-ret quit-q jump-j:
(or, if the end of the list fits)
cont-ret jump-j:

```

Enter **q** to quit the data, **j** to jump to a new type index — with the prompt **index (111-1):**, or **return** to see more of a long list or to quit the data if the end of the list is visible.

### 31.7.5.2 Attractor histogram data decode

The column headings in section 31.7.5.1 and the labels in section 31.7.6 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.
trans ... the average transient (or run-in) length.

```

### 31.7.5.3 Histogram data for density rules

*for binary systems ( $v=2$ ) only*

Enter **a** in section 31.7.1 to include extra data on initial state densities that fall into the all 0s or all 1s point attractors,

An additional column, headed **den-/50%/+**, shows the percentage of initial states with densities ( $d < 0.5$ ,  $d = 0.5$ ,  $d > 0.5$ ) arriving at the all 0s and all 1s point attractors, and **e=** the % error in density classification. For example, the data in figure 31.16 is as follows,

```

attractor types so far=3 sample=2615
type  no    freq    period  den-/50%/+  trans
1     1551  0.5931  1(all 0s)  1091/102/358 e=23%  108.4
2     1054  0.4031  1(all 1s)  130/40/884 e=12%   118.4
3     10    0.0038  2         100/0/0 e=0%      97.2
cont-ret jump-j:

```

### 31.7.6 Print/Save attractor state data

Enter **p** or **S** in section 31.7.3 to print to the terminal, or to save, attractor state data. If saving (to a **.dat** ASCII file) further prompts will appear to set the file name (section 35.3). For 1d networks, the data includes a list of states for each attractor, and if transient data was set in 31.7

(**include trans data-t**) a list of transient states, without duplication, is also included for each attractor (figure 31.19).

The example below is some of the data (without transient data) for the sorted histogram in figure 31.20 showing heading information, a summary for each attractor, and the attractor list, whereas figure 31.19 includes the transient data.

For 2d network the list of attractor states is omitted – the data consists of just of the heading information and a summary for each attractor. For example, below is the start of the data for the sorted histogram in figure 31.17,

```
2d net=20x20 v=2 k=5 rand-wiring att-types=111 sample=30001
```

```
att-type 1: freq=13041=0.4347 period=4 trans=54.1
att-type 2: freq=3610=0.1203 period=8 trans=77.0
att-type 3: freq=3095=0.1032 period=6 trans=40.0
...
```

*(the list of attractor types continues)*

```
...
att-type 109: freq=1=0.0000 period=6 trans=41.0
att-type 110: freq=1=0.0000 period=16 trans=16.0
att-type 111: freq=1=0.0000 period=12 trans=27.0
```

To decode the labels in each attractor summary refer to section 31.7.5.2.

### 31.7.7 Sorting the attractor histogram

Enter **s** in section 31.7.3 to sort the histogram. The following top-right prompt is presented,

```
sort by: attperiod-a avtrans-t (frequency-def):
```

Initially the attractor types are ranked in the order they were found, as in figure 31.17 (top). Enter **return** to sort by frequency. An example is shown in figure 31.17 (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.

```
1d net=15 v=2 k=3 local wiring att-types=63 sample=15579
```

```
att-type 1: freq=3646=0.2340 period=330 trans=6.6
010001111110000
111010000001000
000111000011101
101000100100011
01110111110100
100010000001110
... continues
```

*(the list of attractor types and their states continues)*

```
att-type 63: freq=2=0.0001 period=4 trans=1.0
111111000111110
000000101000001
100001111100011
010010000010100
```

att-type 1: freq=1003/2376=0.4221 period=9 av-trans=1.1 k=3

```
110111
011100
110100
111101
000111
001101
011111
110001
010011
transients to att-type 1
011110
110010
100111
101100
001011
010111
101000
111001
110000
011101
000100
001100
000001
000011
110101
100010
010000
001010
```

transients att-type 1: 18 so far, basin=27

att-type 2: freq=993/2376=0.4179 period=9 av-trans=1.1 k=3

```
111000
101001
111011
001110
011010
111110
100011
100110
101111
```

transients to att-type 2

```
001000
011000
101011
001111
011001
101110
110011
010110
000101
000110
100000
100001
100101
111010
111100
010001
010100
000010
```

transients att-type 2: 18 so far, basin=27

att-type 3: freq=380/2376=0.1599 period=1 av-trans=2.0 k=3

```
000000
transients to att-type 3
110110
111111
101010
010101
100100
101101
011011
001001
010010
```

transients att-type 3: 9 so far, basin=10

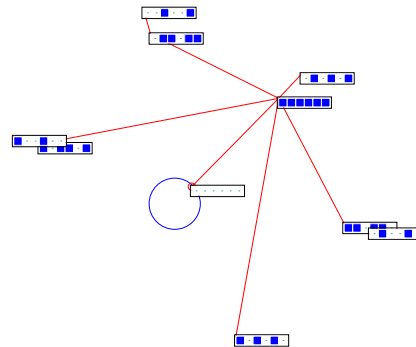
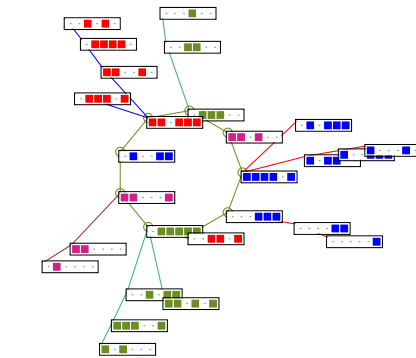
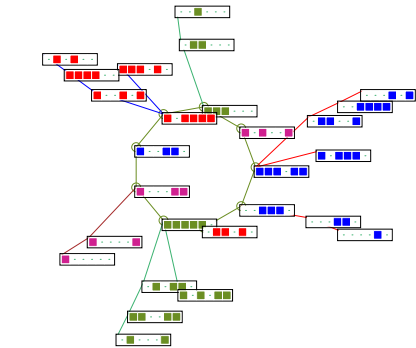


Figure 31.19: If transient data was set in 31.7 (*include trans data=t*) it will be included with attractor data. This example is for a very small CA  $v2k3$  (elementary rule)  $n=6$  rcode 110. The basins of attraction with nodes as bits are inset on the right for comparison (see also figure 24.1). In this case the sample has revealed all attractor states, and all transient states — without duplication. The same type of data can be listed for larger networks, or for RBN and DDN.



### 31.7.8 Attractor histogram jump-graph

The jump-graph (section 20.3) can be applied to the attractor histogram just as to the complete basin of attraction field (section 24.3).

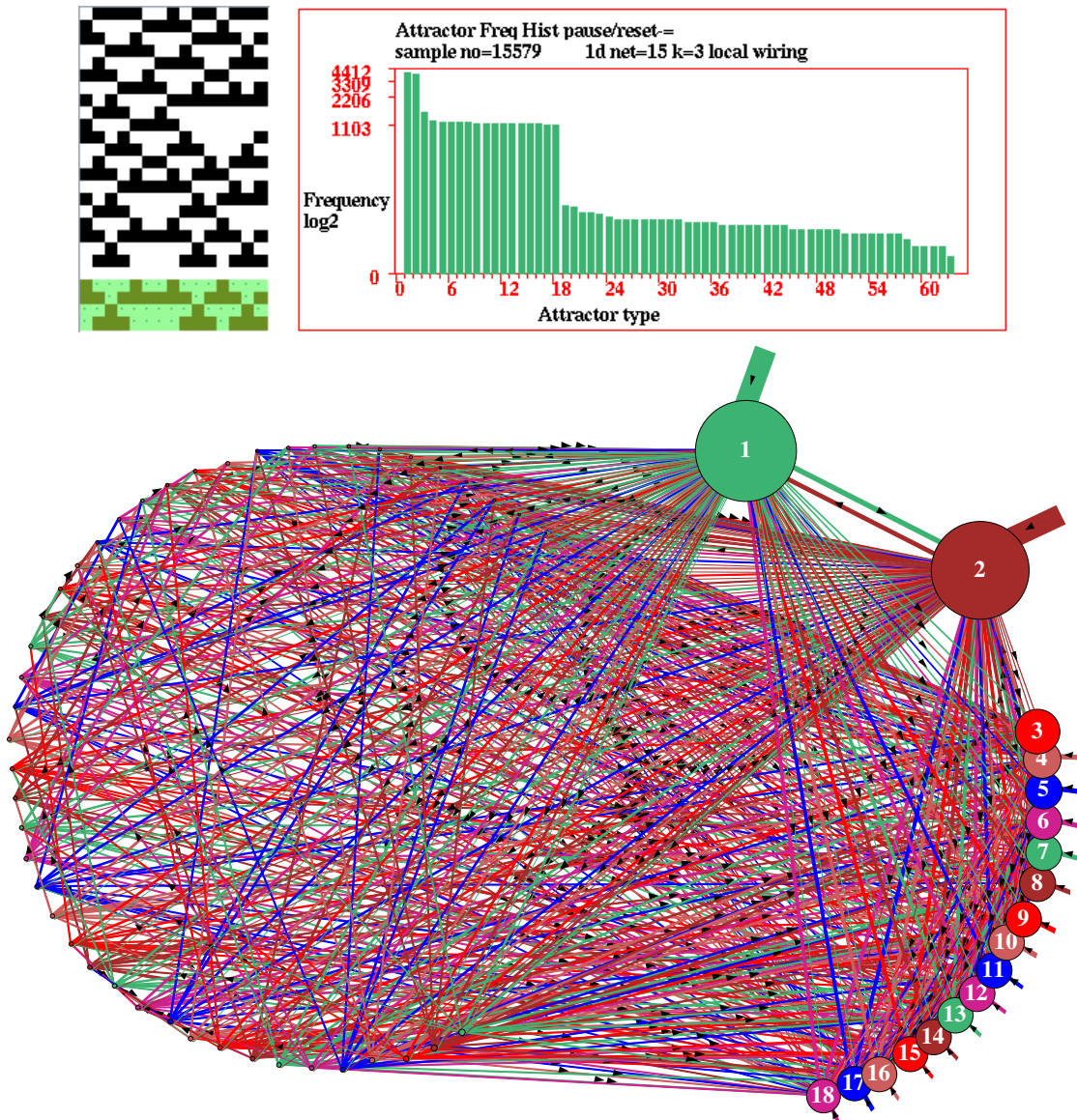


Figure 31.20: The attractor histogram jump-graph, for a 1d CA,  $v3k3$  rcode(dec)54,  $n=15$ . *Top left:* a sample space-time pattern showing the attractor repeated. *Top Right:* The attractor histogram with log  $y$ -axis, all 63 attractors were found after a sample of 15579. *Above:* the jump-graph (chapter 20.3) — nodes and links scaled according to attractor frequency which reflects basin volume, and the graph rearranged 20.5 starting from a circle layout.

Enter **j** in section 31.7.3 to show the jump-graph of the attractor histogram.

Although the example in figure 31.20 took just a few seconds to compute, a jump-graph with many long period attractors would take longer. While the jump-graph is being computed the following top-right message is shown, including an inset box on the right giving the percentage complete (enter **q** to abandon),

**computing jump-graph, quit-q, or wait... 35%**

Once the jump-graph is drawn, it can be rearranged, unravelled, rescaled, including dragging vertices and defined components to new positions with elastic links – all jump-graph functions in chapter 20 apply except redrawing basins at jump-graph nodes 20.7.

## 31.8 Skeleton (fuzzy attractor) histogram

*(for binary systems,  $v=2$ , only)*

The dynamics on large RBN where  $k \geq 3$  typically result in extremely (astronomically) long transients and attractor periods, making it impractical to find attractors by the attractor histogram method (section 31.7), even if a significant fraction of the network is stabilized (frozen) by tuning canalizing inputs (chapter 15), because the rest of the dynamics remains chaotic. To overcome this problem in RBN models of genetic regulatory networks<sup>11</sup> each different frozen sub-pattern is treated as a fuzzy attractor, called the “skeleton” [16, 35]. In this context network elements are referred to as “genes” — though the method can also be applied to CA.

The degree of stability of a frozen sub-pattern to qualify as a skeleton is first defined. The network is run forwards from many random initial states. If the stability threshold is reached, the particular pattern of frozen genes is identified as either a new or previously found skeleton type, and a histogram is built up of the frequency of each type. As for the attractor histogram (section 31.7), various data are recorded (section 31.8.4), including the average transient length.

Enter **s** at the first output parameter prompt (section 31.1) to skip directly to the “skeletons” category of options (the only way to arrive there) to generate the skeleton histogram. The following top-right prompt is presented,

**skeletons: histogram, cancel-q cont-ret:**

Enter **return** to continue and set the parameters to define a skeleton (section 31.8.1).

### 31.8.1 Skeleton parameters prompt

Five parameters are required to define a skeleton. A series of prompts are presented in turn in a top-right window to reset the parameters or accept defaults, as follows,

**frozen skeletons: 0s-0 1s-def both-b:**  
**no of steps frozen: gene (def-20): skeleton (def-20):**  
**% frozen (def-100): % same type (def-100):**

<sup>11</sup>In RBN models of genetic regulatory networks in eukaryotic cells, the different patterns of active genes making cell types in multi-cellular organisms are interpreted as dynamical attractors [20, 25, 36, 40]. RBN attractor skeletons represent the most stable sub-patterns of active genes. The strong overlap between skeletons parallel overlaps in gene expression between eukaryotic cell types [16].

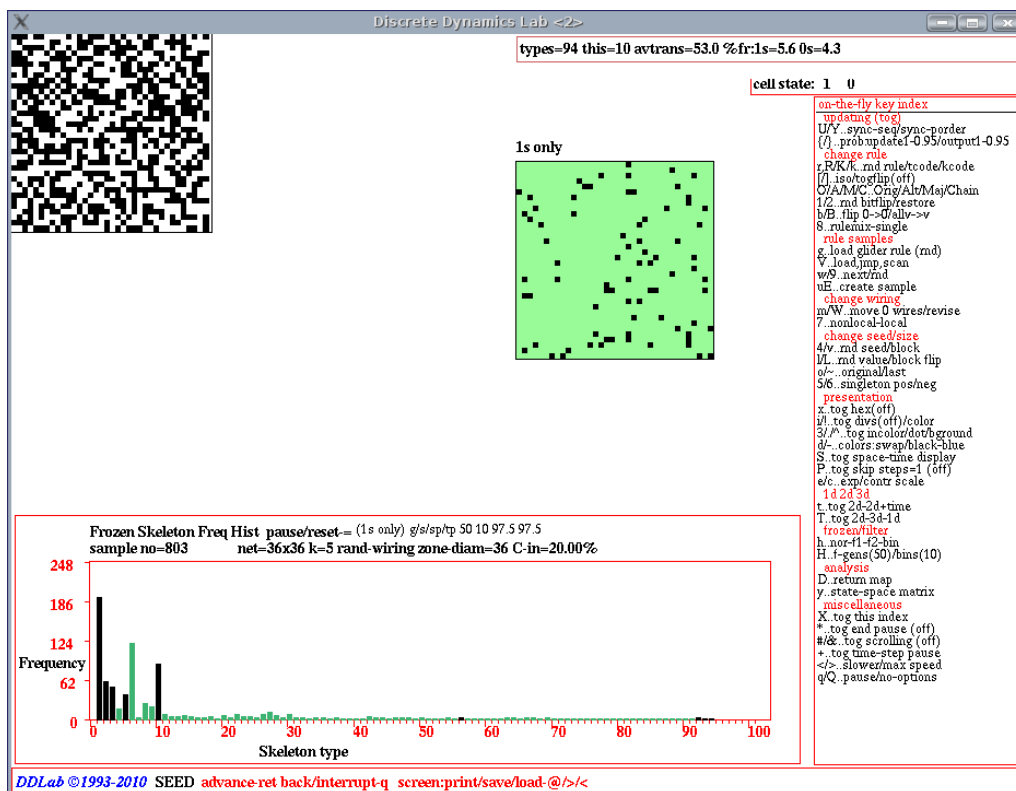


Figure 31.21: A snapshot of the DDLab screen, showing the skeleton histogram at sample 803 with 94 skeletons found.. The network is a  $36 \times 36$   $v2k5$  RBN with unbiased random wiring and 20% canalyzing inputs (chapter 15). The parameters in section 31.8 (1s only), labelled  $g/s/sp/tp$ , are 50, 10, 97.5%, 97.5%. Each sample starts with a new random initial state, and the skeleton into which the transient falls is identified (if any). A skeleton that cannot be matched with a pre-existing type is added as a new type. The histogram columns are continuously updated (in black), showing the skeleton type ( $x$  axis) against the frequency of arriving at that type ( $y$  axis). The axes rescale automatically (and columns turn green) on reaching the current scale limits. *Top left:* a snapshot of the space-time pattern. *Top center:* the skeleton, pattern of frozen 1s. *Bottom:* the unsorted histogram.

The snapshot was grabbed without pausing, but by slowing down updating with on-the-fly option  $<$  (section 32.2).

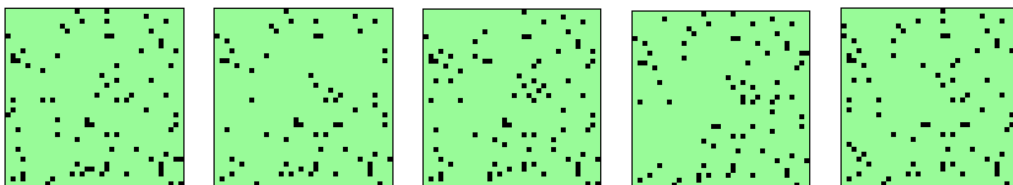


Figure 31.22: Examples of different skeletons, consisting of frozen 1s, from the histogram in figure 31.21. The current skeleton is shown in the top-center of the screen as the histogram is updated.

*options ... what they mean*

- 0s-0 1s-def both-b:** ... The frozen pattern value. Enter **return** for **1s** only, **0** for 0s only, or **b** for both, which is not a good choice if the entire pattern freezes, as in a majority rule.
- gene (def-20)** ... (**g**) The number of time-steps that a gene must remain unchanged to be considered frozen. This is the same parameter as frozen generation size (sections 31.2.3 and 32.11.2).
- skeleton (def-20)** ... (**s**) The number of time-steps the sub-pattern of frozen genes must remain unchanged (within a set Hamming distance) to be considered a frozen skeleton.
- % frozen (def-100)** ... (**sp**) The skeleton percentage Hamming distance limits in (**s**) above.
- % same type (def-100)** ... (**tp**) How close (within a percentage Hamming distance) must a skeleton type be to a pre-established type to qualify as that type, otherwise a new type is established.

The frozen pattern value (0s, 1s or both) and the other parameters, labelled “g/s/sp/tp” as noted above, are also shown in the histogram window (section 31.8.2).

### 31.8.2 Drawing the Skeleton histogram

Just like the attractor histogram (section 31.7.2), the skeleton histogram is drawn in a lower-center window. Space-time patterns (in 1d or 2d) appear in the top-left as usual, from each new initial state. The histogram is continuously updated (in black), 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 the quasi attractor. The axes rescale automatically (and columns turn green) on reaching the current scale limits.

As the histogram is updated, for each sample 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 (figures 31.21 and 31.22). At the same time, a top-right data window gives current data, the current total of skeletons found, the current type, and average transient length, and the percentage of frozen 1s and 0s, for example,

```
types=83 this=4 avtrans=52.9 %fr:1s=5.4 0s=4.8
```

Just as for the attractor histogram, information is displayed at the top of the skeleton histogram window (figure 31.23, decoded in section 31.7.2.1). Skeletons include the following extra items,

```
info ... decode
(1s only) ... or (0s only) or (0s & 1s) — the skeleton frozen pattern value.
g/s/sp/tp ... other skeleton parameters, as listed in section 31.8.
```

Most space-time pattern on-the-fly options work during the histogram routine. To speed the skeleton histogram, toggle space-time graphics off with on-the-fly option **S** (section 32.9.8).

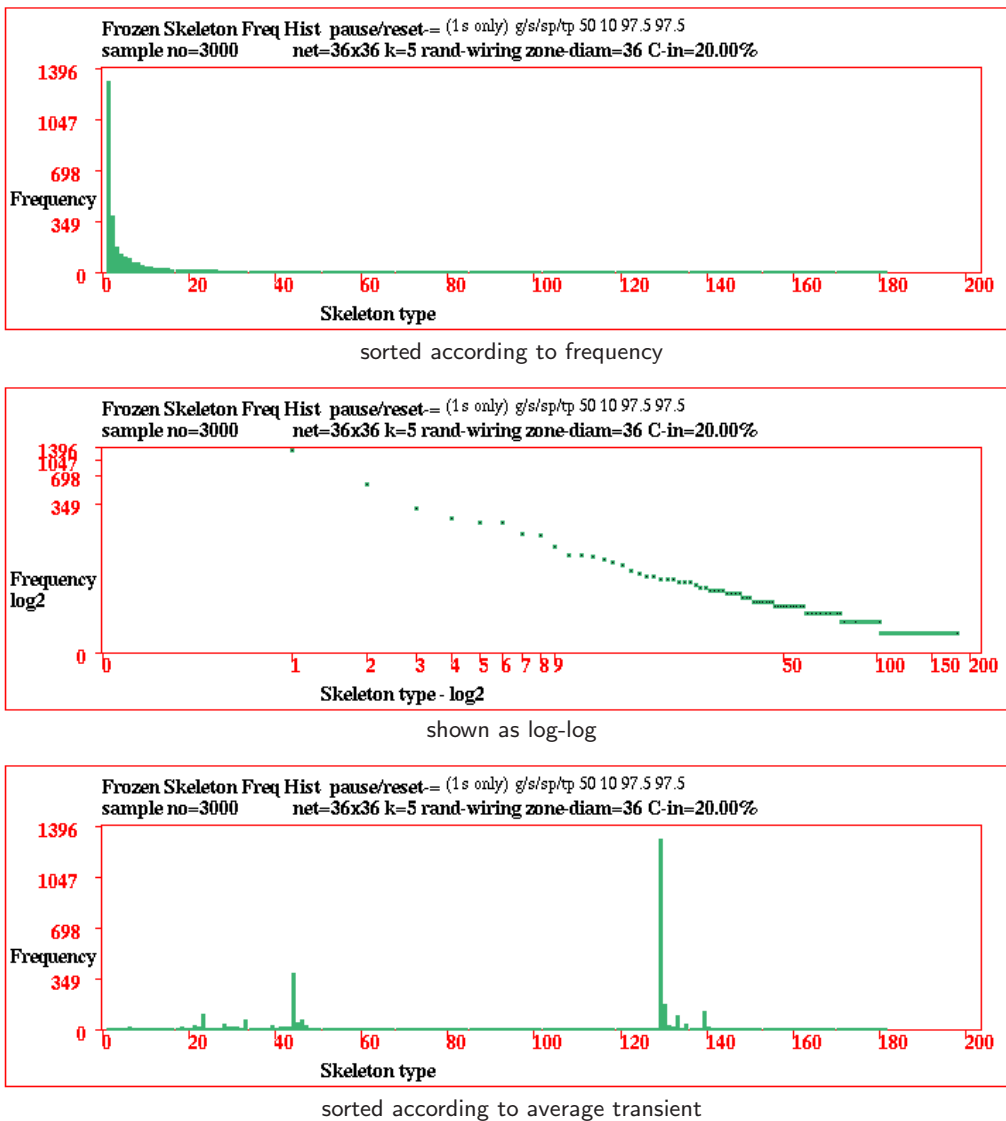


Figure 31.23: The skeleton histogram for the same network as in figure 31.21 but paused at sample 3000 with 181 skeletons found, showing different presentations.

### 31.8.3 Pausing the skeleton histogram

The space-time pattern on-the-fly pause (key **q**, section 32.14) applies while drawing the histogram.

However, there is a special pause for the histogram itself, `enter =` (the equals sign) for the following top-right prompt which provides a range of histogram options,

```
skeletons histogram: hist-h stp-q data-d
redraw-r sort-s undo-pause-u next-ret:
```

These options are summarized below, most are the same as for the attractors histogram (section 31.7.3) — some are described further in the sections indicated,

*options ... what they mean*

**hist-h** ... further histogram options: to save/load, rescale, and log-plot — the same as in “Rescaling the attractor histogram” (section 31.7.4).

**stp-q** ... for the space-time pause prompt with many options (section 32.14. Enter **q** again to backtrack, or **return** to the histogram pause prompt.

**data-d** ... show attractor data in the DDLab screen (section 31.8.4.

**redraw-r** ... to redraw the histogram, which may have been overwritten by other graphics.

**sort-s** ... sort the histogram by frequency and other measures (section 31.8.5).

**undo-pause-u** ... to continue the histogram without pausing.

**next-ret** ... enter **return** to continue the histogram, but pausing after each sample.

### 31.8.4 Skeleton histogram data

Enter **d** in section 31.8.3 to show a list of skeleton types, as many as will fit, displayed in a window on the right of the screen, followed by further options. The list order depends on whether the list was sorted (as in section 31.8.5), if not, the list is ranked in the order that attractor types were found.

The following example is for the sorted histogram in figure 31.17,

```
1s skeleton types so far=181 sample=3000
type  no   freq  %fr-0s-1s  trans
1     1302  0.4340  5.5   5.9  53.0
2     385   0.1283  5.0   5.1  53.1
3     157   0.0557  4.8   5.5  53.0
... (the list continues)
179   1     0.0003  5.1   5.2  52.0
180   1     0.0003  4.5   5.6  52.0
181   1     0.0003  4.3   5.5  52.0
more-ret quit-q jump-j:
(or, if the end of the list fits)
cont-ret jump-j:
```

Enter **q** to quit the data, **j** to jump to a new type index — with the prompt **index (181-1):**, or **return** to see more of a long list or to quit the data if the end of the list is visible.

### 31.8.4.1 Skeleton histogram data decode

The column headings in section 31.8.4 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.
trans ... the average transient (or run-in) length.

```

### 31.8.5 Sorting the skeleton histogram

Enter **s** in section 31.8.3 to sort the histogram. The following top-right prompt is presented,

```
sort by: frozen-0-1-b avtrans-t frequency-def:
```

Initially the skeleton types are ranked in the order they were found, as in figure 31.21. Enter **return** to sort by frequency. as in figure 31.23. This corresponds to sorting according to the fraction of state-space “drained” by the skeleton. Enter **t** to sort by average transient (run-in) length. The histogram can also be sorted by the number of frozen 0s, 1s or both 0s and 1s, enter **0**, **1** or **b**. The result of the new sorting depends on how the list is currently sorted. Having first sorted by frequency, the examples in figure 31.23 show the data sorted first by the attractor period, then by the average transient.

---



## Chapter 32

# Drawing space-time patterns, and changes on-the-fly

Following the output parameters in chapter 31, space-time patterns<sup>1</sup> will start in the top-left of the DDLab screen. This chapter describes parameters and options (including many from chapter 31) that can be reset or activated while space-time patterns are running, either immediately on-the-fly by various key hits, or by pausing the drawing and responding to further prompts.

On-the-fly options include changes to updating, rules, wiring, seed, scale, dimensions, presentation, “frozen” regions, filtering, and various methods of analysis. There are also options for loading glider rules, and automatically classifying rule-space (chapter 33).

If space-time patterns are paused/interrupted (section 32.16), pause options include rules (chapter 16), network architecture (chapter 17), vector PostScript (chapter 36), canalizing (chapter 15), the state (chapter 21), and the network-graph (chapter 20) — to run a concurrent space-time pattern according to the network-graph presentation. On-the-fly options can also be activated during the pause. Many relevant changes and functions in the DDLab manual can therefore be implemented without backtracking.

---

### 32.1 On-the-fly key index

An “on-the-fly key index” window is usually displayed while space-time patterns are running, as a reminder of the various options which are activated directly by key hits. The window appears on the right of the screen and can be toggled on-off the **X** key.

The items listed are context dependent — an example is shown in figure 32.1, and in DDLab screen snapshots in chapters 31, 32 and 33. The options are divided into categories, with sub-headings in red, and may vary for different types of network and current settings. These include the dimensions of the underlying network, and STP-dimensions — the dimensions of the space-time pattern presentation, which are not necessarily the same. On-the-fly options are summarized in section 32.3 and explained in greater detail in sections 32.4—32.13.6.

Note that the same options can be activated during a pause (sections 32.16, 32.17).

---

<sup>1</sup>In TFO-mode, running forward to draw space-time patterns is the only possibility, otherwise this needs to be selected, because the default is to draw attractor basins. If not in TFO-mode, the following steps are required: Enter **s** in section 6.2 for SEED-mode. Then for “space-time patters only”, enter **s** in section 24.1, or **S** in section 29.1



Figure 32.1: An example of the “on-the-fly key index” for space-time patterns — a reminder of the options which can be activated directly by key hits.

The window appears on the right of the DDLab screen and can be toggled on-off with the **X** key.

In TFO-mode the top heading reads

**on-the-fly key index: TFO kcode**

otherwise the heading in SEED-mode is as shown.

The options are divided into subcategories, with sub-headings in red. They are context dependent, and may vary for different types of network, current settings, network dimensions, and STP-dimensions, so some options may be missing or different in an actual run of space-time patterns. This example is for SEED-mode,  $v=2$ , a 1d network with local wiring, presented in 1d (1d-STP), with input-entropy active. Because of this setup, the options below are not included.

#### change rule

8..rulemix/single (*if mixed-rule network*)

#### presentation

@..tog balls outline (*if “balls” or network-graph active*)

d/-..tog shuffle colors/restore (*for  $v \geq 3$* )

#### 1d 2d 3d

t..tog 2d-2d+time (*if 2d-STP*)

p/I..plane/balls (*if 2d+time only*)

p/I/J..plane/balls/invisible (*for 2d-STP scrolling diagonally*)

#### analysis

=..tog diff(keep) (*if “damage” is active, section 31.6*)

#### miscellaneous

\$.make sound (*for DOS only*)

#### Note on scrolling (*section 32.13.3*)

The hash key, #, to toggle scrolling, may not give # on some keyboard settings (noticed in DOS) — in this case the “pounds” key, £, will probably give #, so try using £ to toggle scrolling instead.

#### on-the-fly key index

##### updating (tog)

U/Y..sync-seq/sync-porder  
{/}..prob:update1-0.95/output1-0.95

##### change rule

r,R/K/k..rnd/vrnd/tcode/kcode  
[/.iso/togflip(off)  
O/A/M/C..Orig/Alt/Maj/Chain  
1/2..rnd bitflip/restore  
Z/z..force Z higher/lower  
b/B..flip all0s->0/allVs->V

##### rule samples

g..load glider rule (rnd)  
V..load,jmp,scan  
w/;/9..next/prev/rnd  
uE..create sample

##### change wiring

m/W..move1 wire/revise  
7/|..nonlocal-local/periodic-null

##### change seed/size

4/v/\//..rnd seed/block/vseed/vblock  
l/L..rnd value/block flip  
o/~..original/last  
5/6..singleton pos/neg  
N/n..inc/decrease 1 cell

##### presentation

x..tog slant(off)  
i/!..tog divs(off)/color  
3/./^..tog incolor/dot/bground  
d/-..colors:swap/black-blue  
S..tog space-time display  
P..tog skip steps=1 (off)  
e/c..exp/contr scale

##### 1d 2d 3d

T..tog 1d-2d-3d

##### frozen/filter

h..nor-dy-f1-f2-bin  
H..f-gens(20)/bins(10)  
f/F/a..filter/undo/all

##### analysis

0/%..tog lookhist:1-2/1-time  
)/(..lookhist: amplify/restore  
s..tog entropy-density  
j..tog ent-in-both  
u..entropy/density plot  
G..a-gens (now 10)  
D/;.return map value/density  
y..state-space matrix

##### miscellaneous

X..tog this index  
#.. tog scrolling (on)  
+.. tog time-step pause  
</>..slower/max speed  
q..pause

---

## 32.2 On-the-fly prompts — bottom title bar

The following on-the-fly prompts appear in the bottom title bar (SEED or TFO mode, section 5.5) whenever space-time patterns are running — they are also displayed among the prompts in the on-the-fly key index (figure 32.1, section 32.3).

**STP:tog/exp/contr-S/e/c, tog index-X, slow/max speed:</>**

*DDLab ©1993-2016 SEED advance-ret back/interrupt-q STP:tog/scroll/exp/contr-S/e/c, tog index-X, slow/max-</>*

The key hits are described below,

on-the-fly key hits ... what they means

**STP:tog-S**... to toggle space-time patterns *on* and *off*. If space-time patterns are *off* they continue to run in the background — this considerably speeds up the “damage”, “attractor” and “skeleton” histograms (sections 31.6.2, 31.7, 31.8), and “classifying rule space” (chapter 33).

**STP: -e/c**... to expand or contract the scale of space-time patterns.

**slow/max:</>** ... slow down with < which halves computation speed each time its hit. Enter > to revert to maximum speed. Slow motion can also be invoked at various other stages in DDLab, listed in section 31.2.8.

---

## 32.3 Summary of on-the-fly options for space-time patterns

On-the-fly key index options (section 32.1) are summarized below, divided into categories as in figure 32.1. Further details are given in the sections indicated,

---

updating (tog) — section 32.4

**U/Y..sync-seq/sync-porder** ... sequential and partial order updating (sections 31.4.2 – 31.4.3).

**U** ... toggle synchronous-sequential updating (section 32.4.1)

**Y** ... toggle synchronous-partial order updating (section 32.4.2).

**{/}..prob:update1-0.95/output1-0.95** ... noisy updating with probability settings (section 31.4.1).

**{** ... toggle deterministic-update probability (section 32.4).

**}** ... toggle deterministic-output probability (section 32.4).

---

*change rule* — section 32.5

**r,R/K/k..rnd/vrnd/tcode/kcode** ... new random rule/s — preset density-bias ( $\lambda$  parameter for rcode) is respected (section 16.3.1)

**r** ... for a random rule biased according to the  $\lambda$  parameter, (section 16.3.1), but with the all-0s neighborhood output set to zero for a stable background.

**R** ... for a random rule according to value-bias section 16.3.2.

**K** ... (*SEED-mode only*) random tcode expressed as rcode (t-rcode).

**k** ... (*SEED-mode only*) random kcode expressed as rcode (k-rcode).

**[/].iso/togflip(on)** ... (*rcode only*) new isotropic rcode (section 32.5.2).

[ ... set new isotropic rcode.

] ... tog isotropic bit/value flip (on)/(off). If **togflip(on)** isotropic dynamics is conserved when using on-the-fly **1/2..bitflip/restore** (section 32.5.4), whereas **togflip(off)** does not.

**O/A/M/C..Orig/Alt/Maj/Chain** ... new rule/s (section 32.5.3).

**O** ... restore original rule.

**A** ... random Altenberg rule.

**M** ... random majority rule.

**C** ... random chain rule.

**1/2..rnd bitflip/restore** ... mutate/restore rule/s (section 32.5.4).

**1** ... mutate rule-table by 1 random bit.

**2** ... restore mutated bits in reverse order.

**Z/z..force Z higher/lower** ... (*for single rcode only*) change rule's Z-parameter (section 32.5.5).

**Z** ... force Z higher.

**z** ... force Z lower.

**b/B..flip all0s->0/allVs->V** ... change the output of uniform neighborhoods (the “hot bits” [24]) to the neighborhood value (section 32.5.6).

**b** ... flip output of all the all-0s neighborhood to 0.

**B** ... flip the output of all uniform neighborhoods to the neighborhood value.

**8..rulemix-single** ... (*mixed rule network only, but not k-mix*) toggle rulemix/single rule at cell 0 (section 32.5.7).

---

*rule samples* — section 32.6

**g..load glider rule (rnd)** ... load a glider rule from a collection (if available), either randomly (**rnd**) or sequentially (**seq**) (section 32.6.1).

**V..load,jmp,scan** ... load rule sample, select start, and type of scan. (section 32.6.2 and chapter 33).

- w/:/9..next/prev/rnd** ... scan rules from an active sample.
- w** ... select the next index in the sample or block.
  - :** ... (colon) select the previous index in the sample or block.
  - 9** ... jump to a random sample index.
- uE..create sample** ... enter **u** followed by **E** to create a sample of classified rule-space (section 33.2 and chapter 33). **u** initiates an entropy/density scatter plot (section 32.12.5) — if this is already current, the on-the-fly prompt is **E..create sample**.

*change wiring* — section 32.7

- m/W..move 1 wires** ... move  $x$  wires, the current  $x$  is shown (section 32.7.1).
- m** ... randomly move (rewire) one or more wires.
  - W** ... change the number of wires to move with **m**, or set local (CA) wiring, (1d, 2d, or 3d) depending on native network dimensions.
- 7/|..nonlocal-local/periodic-null** ... nonlocal and local wiring (section 32.7.1.1, or periodic and null boundary conditions (section 32.7.2).
- 7** ... toggle nonlocal and *as if* wiring is local. This does not change any original nonlocal wiring.
  - | ... (pipe or vertical bar) toggle periodic and null boundary conditions (NBC). NBC is described in section 2.7.

*change seed/size* — section 32.8

- 4/v/\//..rnd seed/block/vseed/vblock** ... random seed or block (section 32.8.1) — the preset density-bias (section 21.3.2), or the preset value-bias (section 21.3.3) for a vseed or vblock, and the preset size of the block (section 21.3) are respected.
- 4** ... random seed according to density-bias.
  - k** ... random block according to density-bias.
  - \** ... random seed according to value-bias (vseed).
  - /** ... random block according to value-bias (vblock).
- l/L..rnd value/block flip** ... random value or block (section 32.8.2).
- l** ... mutate the current state by one random bit/value flip — one cell value randomly changed at a random position.
  - L** ... flip a central block randomly in the current state. The block size (section 21.3) and pre-set density-bias (section 21.3.2) are respected.
- o/~..original/last** ... restore previous seeds (section 32.8.3).
- o** ... restore the original seed (chapter 21).
  - ~** ... restore the latest seed that was changed on-the-fly.
- 5/6..singleton pos/neg** ... (for binary  $v=2$ ) set a singleton seed, a central cell against a uniform background. (section 32.8.4).
- 5** ... positive singleton seed, a one against zeros.
  - 6** ... negative singleton seed. a zero against ones.

**5/6..singleton zero/rnd** ... (for  $v \geq 3$ ) set a singleton seed, a central cell against a uniform background. (section 32.8.5).

**5** ... a random value against a uniform background of zeros.

**6** ... a random value against a uniform background of a different random value.

**N/n.inc/decrease 1 cell** ... (for 1d single-rule networks presented in 1d) change network size (section 32.8.6).

**N** ... increase network size by 1 cell.

**n** ... decrease network size by 1 cell.

*presentation* — section 32.9

**x..tog slant(off)** ... (for 1d) a 3-way toggle to slant time-steps (off)-(on-right)-(on-left) — to make asymmetric space-time patterns appear symmetric (section 32.9.1).

**x..tog hex(off)** ... (for 2d) toggle between a square and hexagonal lattice (section 32.9.2).

**@..tog balls outline** ... (if “balls” (section 32.10.3) or network-graph layout (section 32.19) is active, toggle the balls outline (section 32.9.4).

**i/!..tog divs(off)/color** ... (for 1d and 2d STP — takes effect if the cell scale  $\geq 4$ ) to change division lines between cells (section 32.9.3).

**i** ... toggle divisions on-off.

**!** ... toggles division colors black-white.

**3/./^..incolor/dot/bgground** ... for colors by neighborhood, and the presentation of zero cells (section 32.9.5).

**3** ... toggles between cell color by value and by neighborhood.

**.** ... (full stop) (for 1d and 2d STP — takes effect if the cell scale  $\geq 4$ ) toggles having a dot on zero cells.

**^** ... (for 1d and 2d STP, and value colors) toggles between white and light green for zero cells.

**d/-..colors:swap/black-blue** ... (for binary,  $v=2$ ) to swap or change color (section 32.9.6).

**d** ... swaps the color of 0/1 cells between white/black.

**-** ... (minus sign) toggles colors of 1s between black and blue.

**d/-..tog shuffle colors/restore** ... (for  $v \geq 3$ ) to shuffle or restore colors (section 32.9.7).

**d** ... shuffle the colors at random.

**-** ... (minus sign) restore the default colors.

**S..tog space-time display** ... toggle the display of space-time graphics on/off (section 32.9.8).

**P..tog skip steps=1 (off)** ... toggle skipping time-steps (section 32.9.9).

**\$..tog sound** ... (DOS only) toggle sound generated by space-time patterns (section 32.9.10).

**e/c..expand/contr scale** ... for normal space-time patterns: expand/contract the cell scale. For diagonally scrolling network-graph space-time patterns (figures 32.15, 32.17): expand/contract the spacing between time-steps (section 32.9.11).

- e ... expand cell scale (or the spacing) by 1 pixel.
- c ... contract cell scale (or the spacing) by 1 pixel.

1d 2d 3d — section 32.10

- T..tog 1d-2d-3d** ... a 3-way toggle between 1d-2d-3d STP — the order changes to show which is current and which is next in the toggle sequence, e.g. **T..tog 2d-3d-1d**. (section 32.10.1).
- t..tog 2d-2d+time** ... (for 2d STP) toggle between 2d and 2d+time the order changes to show which is current, e.g. **t..tog 2d+time-2d** (section 32.10.2).
- I..tog balls** ... (for 3d STP) toggle between balls and parallelograms (section 32.10.3).
- p/I..plane/balls** ... (for 2d+time STP)
  - p** ... draw 2d plane — parallelogram grid (section 32.10.4).
  - I** ... toggle cells between balls and parallelograms (section 32.10.3).
- p/I/J..plane/balls/invisible** ... (for 2d STP diagonal scrolling)
  - p** ... draw 2d plane — square or hex grid (section 32.10.4).
  - I** ... toggle cells between balls and squares/hexagons (section 32.10.3).
  - J** ... toggle the visibility of 2d diagonal scrolling STP to see just the scrolling network-graph (sections 32.10.5, 32.13.3.3, figure 32.15).

frozen/filter — section 32.11 (if input-entropy is active) filter (otherwise)

- h..nor-dy-f1-f2-bin** ... 5-way toggle to show, dynamic trace, and “frozen” cells in 3 different ways (section 32.11.1).
- H..f-gens (now 20)/bins(10)** ... change “frozen” parameters (section 32.11.2).
- f/F/a..filter/undo/all** ... (if input-entropy active) (section 32.11.5)
  - f** ... progressively filter.
  - F** ... progressively unfilter in reverse order.
  - a** ... restore all filtering.

analysis — section 32.12

- s..tog entropy-density** ... (if input-entropy is active) toggle input-entropy/pattern density (section 32.12.3).

**0/%..tog lookphist:1-2/1-time** ... (if *input-entropy* is active — section 32.12.1) the presentation of the input-frequency histogram.

**0** ... (*zero key*) toggle between normal and 1 pixel width bars.

**%** ... show with the histogram with a time dimension — if the number of bars  $\leq 64$ .

**)/(..lookhist: amplify/restore** ... (if *input-entropy* is active section 32.12.3) amplify the input-frequency histogram.

) ... double the length of bars each time “)” is pressed.

( ... restore the default length.

**s..tog entropy-density** ... toggle showing the input-entropy or pattern densities (as a set of  $v$  value-density graphs) instead of the input-entropy (section 32.12.3).

**j..tog ent-in-both** ... (if *input-entropy* is active section 32.12.3) a 3-way toggle to show the input-entropy, the input-frequencies as a set of graphs, or both together (section 32.12.4) (section 32.12.5).

**u..entropy/density plot** ... (if *input-entropy* is active section 32.12.3) toggle showing a scatter plot of entropy-density where rules have characteristic signatures. (figure 32.32, section 32.12.5).

**G..a-gens (now 10)** ... change analysis generations (section 32.12.6).

**D/;..return map value/density** ... toggle return map scatter plots either by value, or by density.

**D** ... the return map by value (section 31.2.2.2).

;**;** ... (semi-colon) the return map by density (section 32.12.8) but only if the space-time pattern density (section 32.12.3) is active.

**y..state-space matrix** ... toggle state-space matrix (section 32.12.9).

**=..tog diff (keep damage)** ... (if the “*damage*” is active, section 31.6), toggle between two definitions of damage, (**keep**) — once damaged, keep the damage, and (**inst**) — the instantaneous difference at each time-step.

---

*miscellaneous* — section 32.13

**X..tog this index** ... toggle the on-the-fly key index (section 32.13.1).

**\*..tog end pause (on)** ... toggle the end pause on/off (section 32.13.2).

**#..tog scrolling** ... toggle the scrolling on/off with the “hash” key<sup>2</sup> # (section 32.13.3).

**+..tog time-step pause** ... pause after each time-step (section 32.13.4).

**</> ..slow/max speed** ... space-time pattern speed: < slow down, > restore full speed (section 32.13.5).

**q..pause** ... pause space-time patterns and display data/options (section 32.13.6).

---

<sup>2</sup>The hash key, #, to toggle scrolling, may not give # on some keyboard settings (noticed in DOS) — in this case the “pounds” key, £, will probably give #, so try using £ to toggle scrolling instead.





### 32.5.1 $r, R/K/k..rnd/vrnd/tcode/kcode$

Set a new random rule (or rules for a rulemix) where the density-bias set in section 16.3.1 is respected. Information about the new rule appears in a top-right window similar to figure 32.2, with possible labels: rcode, tcode, kcode, k-rcode, or t-rcode.

$r$  ... for a random rule biased according to the  $\lambda$  parameter, (section 16.3.1), but with the all-0s neighborhood output set to zero for a stable background.

$R$  ... for a random rule according to value-bias section 16.3.2.

$K$  ... (if not in TFO-mode) random tcode expressed as rcode (t-rcode).

$k$  ... (if not in TFO-mode) random kcode expressed as rcode (k-rcode).

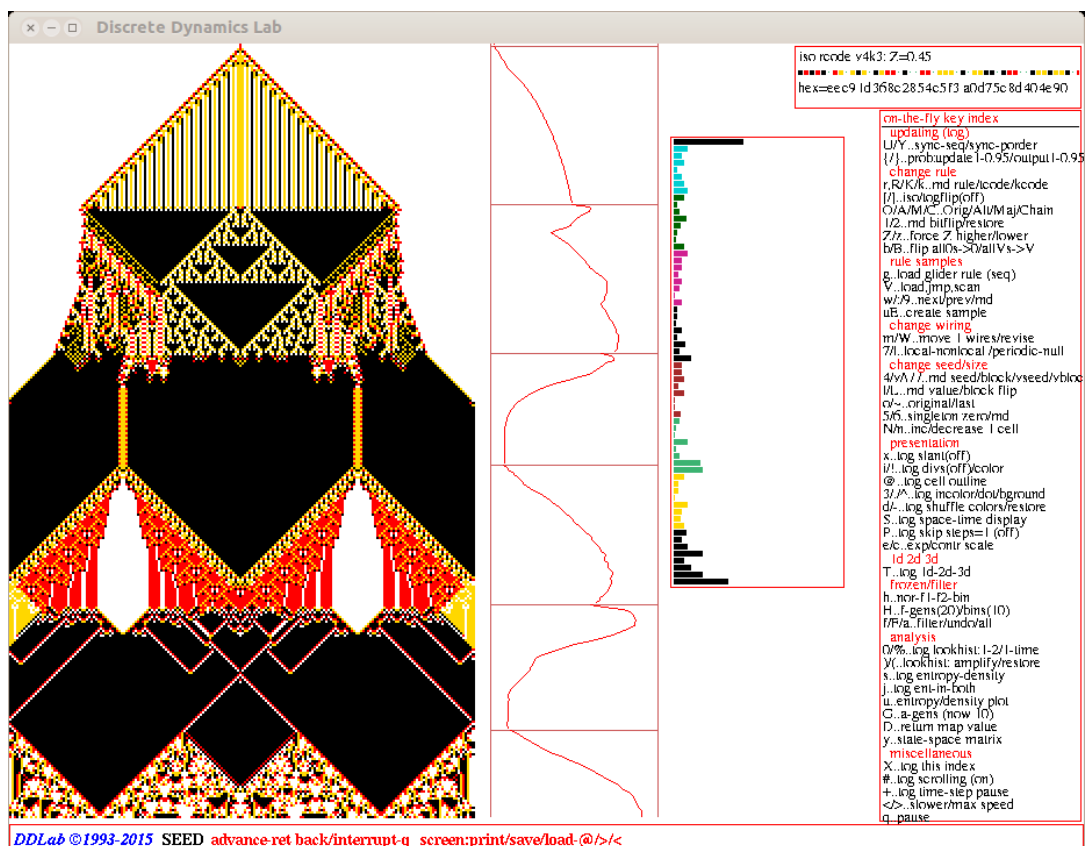


Figure 32.3: A snapshot of the DDLab screen, showing a 1d  $n=200$  space-time pattern made up of a series of  $v4k3$  random isotropic rcodes, where reflected neighborhoods have the same output, set on-the-fly with key [ (left square bracket). Note that the symmetry is preserved. The start of each new rule, which picks up the current state from the previous rule, is indicated by a horizontal line on the entropy plot to the right of the space-time pattern. The last rule set is shown in a top-right window.



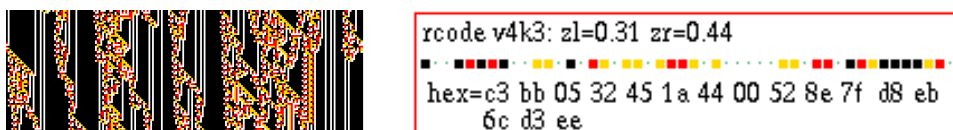


Figure 32.5: Info when changing the  $Z$ -parameter up and down on-the-fly with keys  $Z$  and  $z$ . Initial dynamics with high  $Z$  was progressively changed towards order by repeated hits with key  $z$  — ordered domains emerged. *Left*: about 70 time-steps of the 1d  $n=200$  space-time pattern, and *Right*: information on the new rule that appears in a top-right window when the key is hit, where  $zl$  is  $z_{left}$  and  $zr$  is  $z_{right}$  — the  $Z$ -parameter is the greater of the two[31, 48].

### 32.5.5 $Z/z$ ..force $Z$ higher/lower

*for single rcode only*

The current rule can be adapted by tuning the  $Z$ -parameter, progressively forcing  $Z$  higher (towards chaos) or lower (towards order), by selectively mutating the rcode. This is done on-the-fly with the following key hits,

- Z** ... (upper case) to force  $Z$  higher.
- z** ... (lower case) to force  $Z$  lower.

Information about the rule appears in a top-right window (figure 32.5). The algorithm is the same as in section 16.3 — flipping bits/values at random positions, and only retaining the flips that produce the desired change in  $Z$ . The algorithm for forcing  $Z$  lower can get stuck, but can usually be unstuck with a random bit/value flip (section 32.5.4).

### 32.5.6 **b/B**..flip all0s->0/allVs->V

The “hot bits” in a  $v=2$  rule-table, the outputs of the all-0s and all-1s neighborhoods, are especially important in the dynamics[24]. This notion is extended for  $v \geq 3$ , and the following on-the-fly key hits flip all-0s to 0, or flip the outputs of all uniform neighborhoods to the neighborhood value, which provides a stable background for singleton seeds (section 32.8.4).

- b** ... flip output of all-0s to 0.
- B** ... flip the output of all uniform neighborhoods to the neighborhood value.

Information about the new rule appears in a top-right window similar to figure 32.2, with possible labels: rcode, kcode, tcode. All the rules in a rulemix are modified.

### 32.5.7 **8**..rulemix-single

*mixed rule network only, but not k-mix*

Enter **8** to toggle on-the-fly between running the network according to the its rulemix, 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.

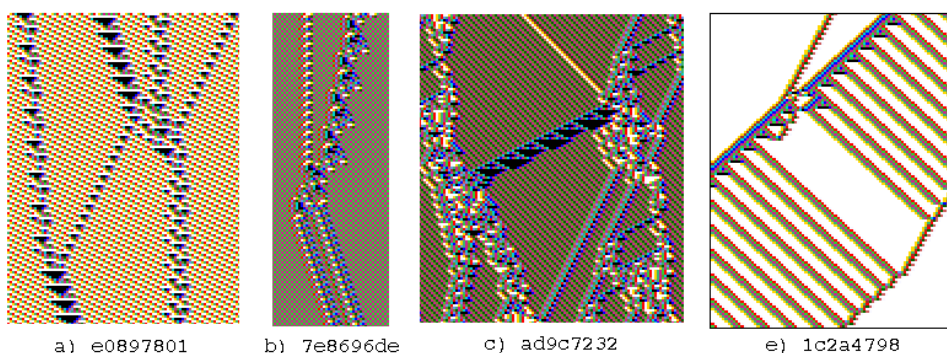


Figure 32.6: Examples of  $v2k5$  complex space-time patterns with interacting gliders[33] from a sample of glider rules (`g_v2k5.r_s`) included with DDLab, which can be loaded on-the-fly while space-time patterns are being drawn (cells colored by neighborhood, section 31.2.1). The rcodes are shown in hex.

## 32.6 Rule samples

These on-the-fly options relate to glider rule collections (section 32.6.1), and to automatically classifying rule-space[38] (sections 32.6.2 — 32.6.4), including creating, sorting, analysing and probing the sample. Automatically classifying rule-space is a larger topic described and illustrated in detail in “Classifying Rule Space” chapter 33.

### 32.6.1 `g..load glider rule (rnd)`

*if a rcode or kcode glider file is detected*

Collections of complex (or “glider”) rules are available for some combinations of  $[v, k]$  and lattice dimensions<sup>3</sup>. The available files are listed in section 3.6.1. If the current  $[v, k]$  corresponds to one of these files, and if the directory/file is accessible (section 35.5), on-the-fly key `g` loads a rule from the collection while space-time patterns are running (figure 32.8), either randomly (`rnd`) or in sequence (`seq`), depending on the setup in section 31.2.9 — this can also be toggled from the pause prompt (section 32.16). On-the-fly key `g` loads a rule, and information is simultaneously shown in a top-right window, including the collection size and sample number (figure 32.9). Otherwise a top-right inset shows `glider sample not available`.

If a rulemix is active (but not a  $k$ -mix) the rule is loaded at cell index 0 — this can be applied to the whole network by toggling with key `8..rulemix-single` (section 32.5.7).

#### 32.6.1.1 creating a glider rule collection

Glider rule collections are ordinary “rulemix-only” files (sections 19.4, 19.2, 19.3) but with a special type of filename, where  $x$  is the value-range, and  $y$  the neighborhood size.

```
g_vxky.r_s ... for rcode, for example g_v3k3.r_s
g_vxky.r_v ... for kcode, for example g_v3k6.r_v
```

<sup>3</sup>Although the complex rules relate to specific dimensions, they often result in interesting dynamics in other than the intended dimensions.

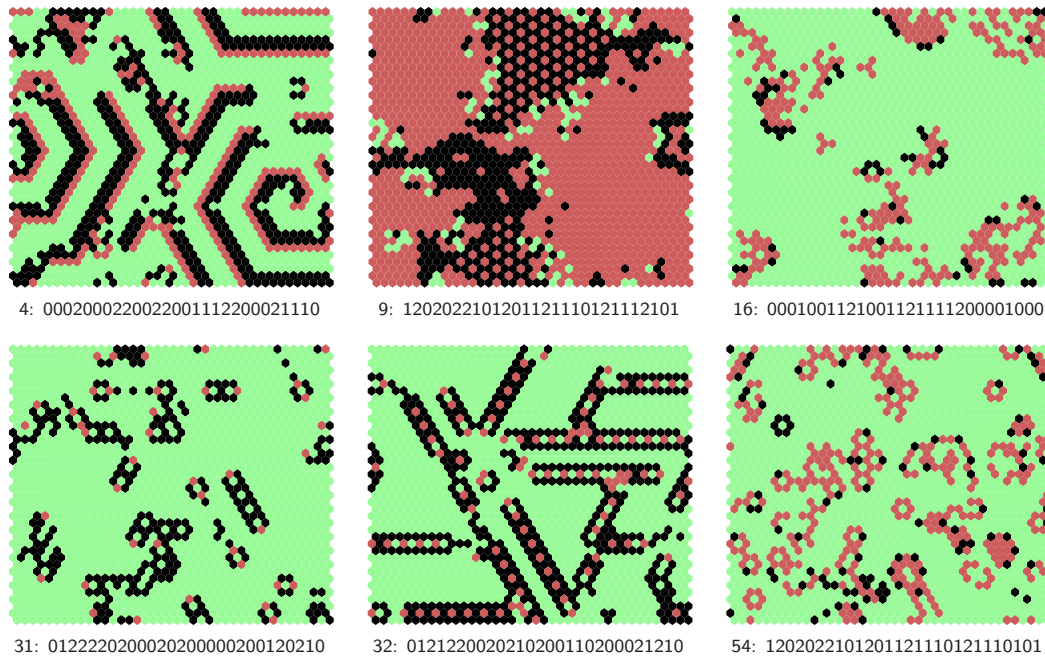


Figure 32.7: Examples of 2d space-time snapshots of complex kcode on a 2d  $40 \times 40$  lattice, from a sample of glider rules (`g_v2k6.r_v`) included with DDLab, which can be loaded on-the-fly. The sample number and kcode table are shown for each rule. The color of zero (background) values were toggled to light-green with on-the-fly key  $\wedge$  (circumflex).

The special name is recognized by DDLab. An available file for the current  $[v, k]$  is automatically loaded after the neighborhood  $k$  is set in chapter 9, also revealing the number of rules in the collection. If a collection is not available for the current  $[v, k]$ , on-the-fly key `g` (section 32.6) will display `glider sample not available` in a top-right inset.

New collections can be created, extended and modified by loading single rule files or `rulemix`-only into another a `rulemix`, which must then be saved with the correct special name. This is best done in a 1d wiring graphic. (section 19.4). If the number of rules to be loaded is  $g$ , the size of the 1d network  $n$  should be set at  $n=g+1$  because index 0 is not referenced, and this needs also to be taken into account when combining two `rulemix`-only files to make a larger glider file.

### 32.6.2 `V..load, jmp, scan`

Automatically classified samples of rule-space (chapter 33), sorted by input-entropy, and its variability[38] (either standard deviation or a min-max measure (section 33.1), can separate ordered, complex and chaotic rules and show their distribution in rule-space.

Some samples files are available for various combinations of  $[v, k]$  and lattice dimensions, listed in section 3.6.1. The files have the extension `.sta`. If a sample is not currently active, on-the-fly key `V` will load, then display, the sample (section 33.5). The sample can also be loaded during a pause (section 32.16.2). If a sample is currently active (i.e.loaded), entering on-the-fly key `V` results in the following top-right prompt,

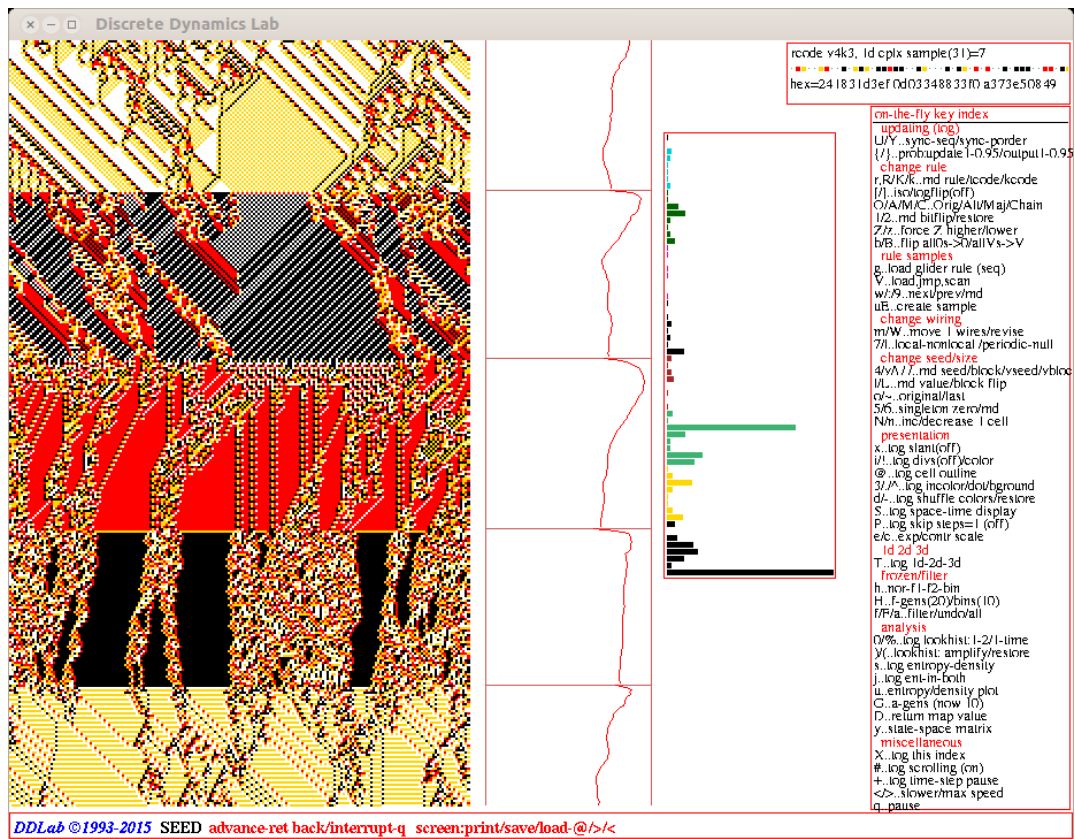


Figure 32.8: A snapshot of the DDLab screen, showing 1d  $n=200$  space-time patterns, a series of “glider” rules from the  $v4k3$  complex rcode collection loaded on-the-fly with key  $g$ . The start of each new rule, which preserves the current state, is indicated by a horizontal line on the entropy plot to the right of the space-time pattern. The last rule set is shown in a top-right window, including its sample number.

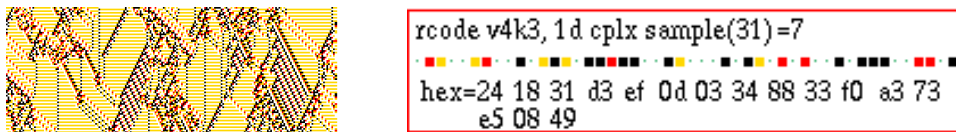


Figure 32.9: Info when loading a complex rule on-the-fly with key  $g$ . *Left*: about 70 time-steps of the 1d  $n=200$  space-time pattern, and *Right*: information on the new rule that appears in a top-right window when the key is hit. “1d cplx sample(31)=7” means that rule number 7 from 31 rules in the  $v4k3$  complex rcode collection is currently selected.

**rule sample: load new-n, rule index/scan 1-15834:** (*kcode file v3k6bs.sta*)

Enter **n** for a new sample, otherwise the space-time patterns of successive rules can be scanned, from rule index 1 (the default, enter **return**) or from any rule index (enter a number) — described in sections 33.7, 33.7.2. For further details see chapter 33.



### 32.6.3 w/:/9..next/prev/rnd

Once a rule sample is active (i.e. loaded, section 32.6.2 above), the sample, or a predefined block (section 33.7.2), can be scanned with the following key hits, to select and immediately apply individual rules,

- w** ... for the next rule index in the sample or block, starting with index 1 or another index set in section 32.6.2.
- :** ... (colon) for the previous index in the sample or block.
- 9** ... to jump to a random index in the sample (the block in this case does not apply).

Section 33.7 and 33.7.2 give further explanations. Each time a rule is changed, a top-right window shows the sample index, the rule's mean-entropy and variability (standard-deviation or min-max), and the rule itself in hex, as in figure 33.14. For further details see chapter 33.

### 32.6.4 uE..create sample

Enter **u** followed by **E** to create a sample of automatically classified rule-space. Prompts are presented as described in sections 33.2. The initial **u** initiates an entropy/density scatter plot (section 32.12.5). If this plot is current, the on-the-fly prompt is **E..create sample**, so enter just **E** to create a sample. For further details see chapter 33.

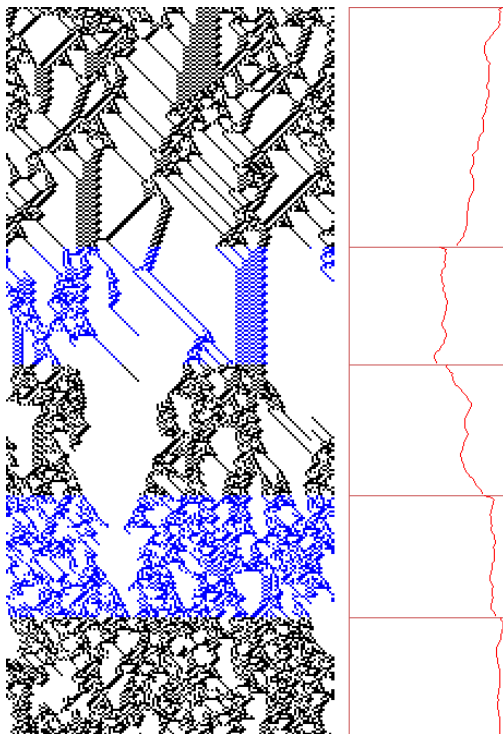


Figure 32.10: Randomizing the wiring of a 1d CA by stages, for  $v2k5$  rcode(hex)6c1e53a8,  $n=150$ . The wire moves were first revised to 2% (15 out of 750 wires) with on-the-fly option **W**, then the wiring was cumulatively randomized with on-the-fly with option **m** (section 32.7.1), at the points indicated by color changes, and horizontal lines in the input-entropy plot. Note that glider structure is progressively degraded.

## 32.7 Change wiring

These on-the-fly options change (mutate) the wiring (section 32.7.1), toggle between local and nonlocal dynamics in a nonlocal network section 32.7.1.1, or toggle between periodic and null boundary conditions (section 32.7.2).

### 32.7.1 m/W..move 1 wires

Enter **m** to randomly move one wire or a preset number of wires (selected at random), which can be repeated for cumulative rewiring (figure 32.6.4).

Enter **W** to reset the number of wires to be randomly moved with **m**, or to localize the wiring. Prompts similar to “mutate wiring” (section 28.2) will be presented in a top-right window, for example,

```
wires to move=1/9600=0.0104%: all-a number-n %-p:
local-L: (for a 2d 40x40 k=6 network)
```

*options ... what they mean*

- all-a** ... to select all wires, i.e. randomly rewiring the whole network when **m** is pressed.
- number-n** ... to reset a given number of wires to move, selected with the following subsequent prompt, ... **number of wires:** — the new setting will show up in the on-the-fly prompt, for example, **m/W..move 25 wires**
- %-p** ... to specify the percentage of total wires to move, selected with the following subsequent prompt, ... **% of total wires:**
- local-L** ... to restore local (CA) wiring (1d, 2d or 3d) depending on native network dimensions. For mixed-*k* each cell is assigned the relevant local neighborhood.

#### 32.7.1.1 7..nonlocal-local

Enter **7** on-the-fly to toggle between running a network with nonlocal wiring, and running the network *as if* the wiring was local (1d, 2d or 3d) depending on native network dimensions — where local (CA) wiring is defined in chapter 10.

The original nonlocal wiring held in memory remains intact when local wiring is toggled. The current status is given by the prompt order, i.e. **..local-nonlocal** indicates that the network is currently being run as if it has local wiring. If the wiring held in memory is itself local, there will be no difference between local and nonlocal. However, wire moves that perturb local wiring with **m** in section 32.7.1 will only show up if nonlocal wiring is toggled.

For mixed-*k* with random wiring, toggling to local wiring treats each cell as having the relevant local neighborhood.

#### 32.7.2 |..periodic-null

Enter **|** (pipe or vertical bar) to toggle between periodic boundary conditions (PBC) and null boundary conditions (NBC). NBC is described in section 2.7. The current status is given by the prompt order, i.e. **..null-periodic** indicates that NBC is current.



## 32.8 Change seed/size

These on-the-fly options amend (mutate) the current state or seed, and also (for 1d networks displayed in 1d) the of size of the network .

### 32.8.1 4/v..rnd seed/block

When setting a random seed or block on-the-fly, the preset density-biases (section 21.3.2), and the size of the block (section 21.3) are respected. Enter key hits as follows,

- 4 ... to reset the current network state at random.
- v ... to reset the network state as a central random block (in 1d, 2d or 3d).  
The size of the block was set in section 21.3.

### 32.8.2 1/L..rnd value/block

To mutate or perturb the current state, enter on-the-fly key hits as follows,

- 1 ... to flip/change one bit/value randomly at a random position.
- L ... to flip a central block randomly in the current state. The block size (section 21.3) and pre-set density-bias (section 21.3.2) are respected.

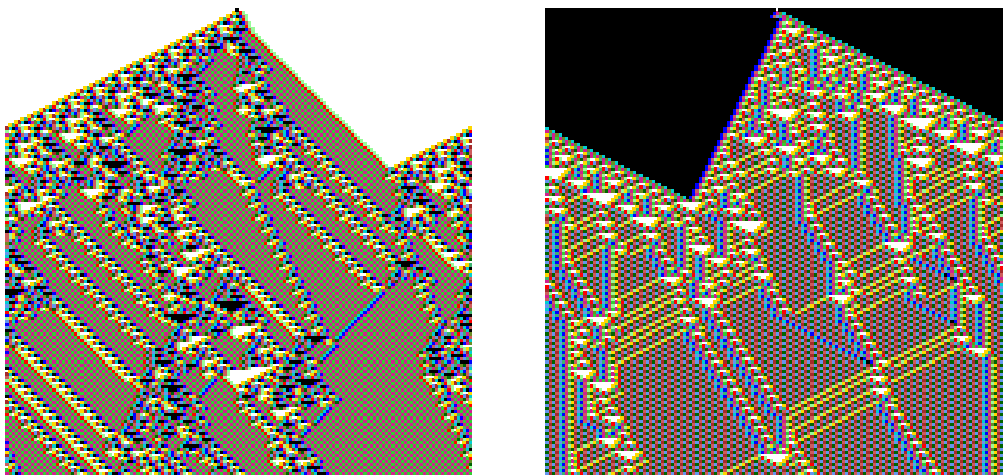


Figure 32.11: *Left*: a positive singleton seed, a central 1 surrounded by 0s, and *Right*: a negative singleton seed, a central 0 surrounded by 1s, set on-the-fly with key hits 5 and 6 (section 32.8.4).  
*Left*:  $v2k5$  rcode(hex) 968c4f76, *Right*:  $v2k5$  rcode(hex) rule d28f022c.  $n=150$ , 150 about time-steps, color by neighborhood (section 32.9.5).

### 32.8.3 o/~..original/last

To restore previous seeds/states, enter on-the-fly hits as follows,

- o ... to restore the original seed (chapter 21).
- ~ ... to restore the latest seed that was changed on-the-fly by any method in sections 32.8.

### 32.8.4 5/6..singleton pos/neg

*for binary  $v=2$*

Enter **5** or **6** on-the-fly to reset the seed as a positive or negative singleton seed, a central 1 surrounded by 0s, or a central 0 surrounded by 1s (figure 32.11).

### 32.8.5 5/6..singleton zero/rnd

*for  $v \geq 3$*

Enter **5** or **6** on-the-fly to reset the seed as a singleton seed, **5** for a random non-zero value against a uniform background of zeros, **6** for a random value against a uniform background of a different random value.

### 32.8.6 N/n..inc/decrease 1 cell

*only for 1d single-rule networks shown in 1d*

Enter **N** or **n** on-the-fly 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 originally 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 example,

**n increased 166 -> 167 or n decreased 167 -> 166.**

## 32.9 Presentation

These on-the-fly options amend various aspects of space-time pattern the presentation, including color and scale.

### 32.9.1 x..tog slant(off)

*for 1d networks only, shown in 1d*

Enter **x** for a 3-way toggle to slant time-steps (off)-(on-right)-(on-left) — shown in the on-the-fly index. This can make asymmetric space-time patterns appear symmetric. If the slant is *on*, successive time-steps are shifted by half a cell-width either right or left as in figure 32.12<sup>4</sup>.

<sup>4</sup>For the vector PostScript image of slanted space-time patterns, an odd number of time-steps should be selected to avoid distortion.

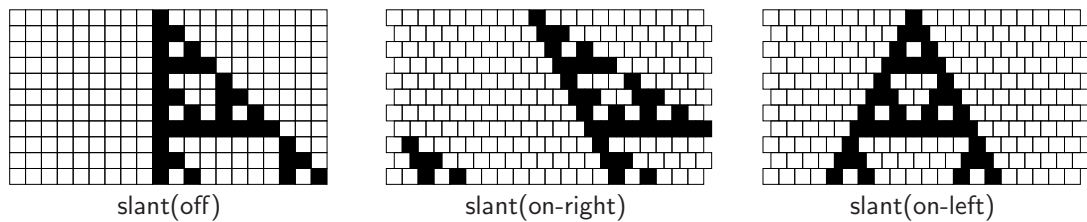


Figure 32.12: Slanting 1d space-time patterns with key `x`, a 3-way toggle (section 32.9.1). If the slant is *on*, successive time-steps are shifted by half a cell-width either right or left. In this example, left slant produces symmetry — for the elementary CA `v2k3 rcode(dec)60`.

### 32.9.2 `x..tog hex(off)`

*for 2d networks only, and 2d STP*

Enter `x` on-the-fly to toggle between a square or hexagonal lattice (figure 32.13). An even number of rows is required for a hexagonal lattice to avoid an inconsistent presentation. The on-the-fly index gives the current status: `hex(off)` or `hex(on)`.

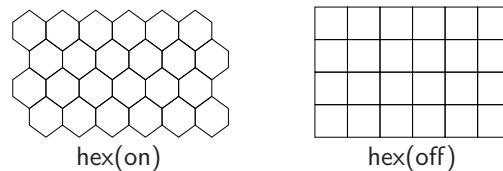


Figure 32.13: Toggling between a hex or square 2d lattice with key `x` (section 32.9.2). A proper hex presentation requires an even number of rows

### 32.9.3 `i/!.tog divs(off)`

*for 1d and 2d STP — takes effect if the cell scale  $\geq 4$*

To set division lines between cells, or change division line color, (figure 32.14) enter on-the-fly key hits as follows,

- `i ...` to toggle divisions on-off. The on-the-fly index gives the current status — `divs(off)` or `divs(on)`.
- `! ...` for a 3-way toggle of the to division color — white, black, and light blue.

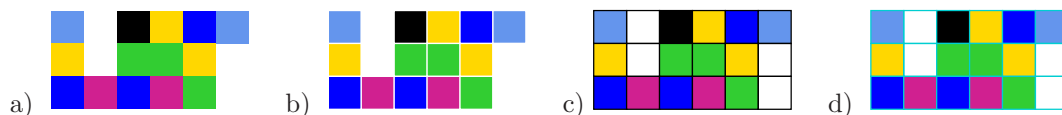


Figure 32.14: Changing the division line presentation/color between cells: a) no lines — cells touch. b) white. c) black. d) light blue. 2d square layout  $6 \times 3$  (for a hex layout see figure 21.7).

### 32.9.4 @..tog balls outline

*if “balls” or network-graph layout is active*

If cells are shown as “balls”, which always applies in a network-graph (section 32.19), enter @ on-the-fly to toggle balls with and without an outline (figure 32.15).

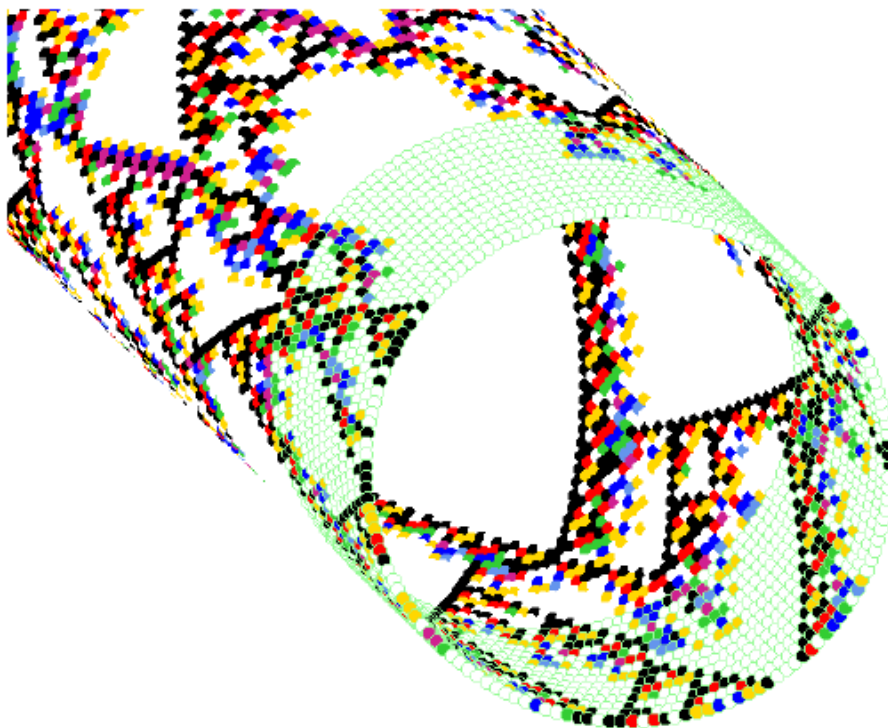


Figure 32.15: Toggling the cell outline in network-graph space-time patterns (section 20.13) with on-the-fly key @. This example is a 1d (scrolling) ring of cells (as figure 4.9). The cell outline is active for the leading 14 time-steps only, showing up the zero-value cells in particular.  $n=150$ ,  $v8k:3$  kcode, index 7 from the rule sample in figure 33.5. Figure 32.37 shows the circle layout without scrolling.

### 32.9.5 3/./^..incolor/dot/bground

On-the-fly key hits for cell colors by neighborhood, and the presentation of zero (background) cells, are as follows,

- 3 ... to toggle between cell colors according to the neighborhood “looked up” and the actual cell value. This can also be preset in section 31.2.1. Examples are shown in figure 32.23 (a,b), and figures 31.1 — 31.3 for 1d, 2d and 3d networks.
- . ... (for 1d and 2d STP) (full stop) toggles putting a dot on zero cells. Dots appear for the cell scale  $\geq 4$  pixels.
- ^ ... (for 1d and 2d STP, and value colors) to toggle between white and light green for zero cells. This is sometime necessary to show up the background.

### 32.9.6 **d/-..colors:swap/black-blue**

*for binary,  $v=2$*

To swap or change color, enter on-the-fly key hits as follows, The color key as in figure 7.1 is displayed top-right.

- d** ... to swap colors of 0/1 cells between white and black.
- ... (minus sign) to toggles the color of 1s between black and blue.

### 32.9.7 **d/-..tog shuffle colors/restore**

*for  $v \geq 3$*

To shuffle or restore colors, enter on-the-fly key hits as follows, The color key as in figure 7.1 is displayed top-right.

- d** ... to randomly shuffle the colors assigned to different values.
- ... (minus sign) to restore the default colors.

### 32.9.8 **S..tog space-time display**

Enter **S** on-the-fly to toggle the display of space-time graphics on/off. Not showing space-time patterns, which continue to run in the background, speeds up other processes, for example creating a sample of automatically classified rule-space (chapter 33), histograms of “damage” (section 31.6.2), and histograms of attractors (section 31.7).

### 32.9.9 **P..tog skip steps=1 (off)**

Enter **P** on-the-fly to toggle skipping on/off. The number of time-steps to be skipped can be changed in section 32.16.8. The default is 1, i.e. showing every second time-step. The current number of time-steps, and whether skipping is currently on or off, is indicated in the prompt, for example, **P..tog skip steps=3 (on)**.

### 32.9.10 **\$.tog sound**

*DOS only*

Enter **\$** on-the-fly to toggle sound generated by space-time patterns. Both the pitch and delay are generated by 8 bits near the center of the network.

### 32.9.11 **e/c..expand/contr scale**

Enter **e** on-the-fly 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 pattern in 1d, 2d or 3d. The initial scale can be preset in section 31.2.4.

Keys **e/c** have a different effect for diagonally scrolling network-graph space-time patterns (section 20.13), as in figure 32.15 — the spacing between time-steps is expanded or contracted by one pixel as illustrated below.

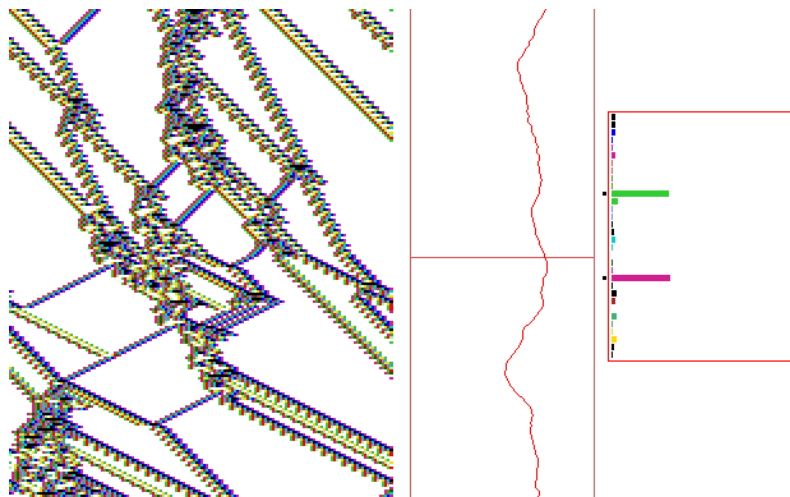


Figure 32.16: Skipping time-steps in a 1d CA with the default `steps=1` (section 32.9.9), activated on-the-fly with key hit **P** at about the mid point of the space-time pattern (after about 100 time-steps) — indicated by a horizontal line in the input-entropy plot — so skipping alternate time-steps thereafter. The space-time pattern is shown according to neighborhood color, and is also filtered — indicated by square dots in the input-frequency histogram. `v2k5 rcode(hex)968c4f76`,  $n=150$ .

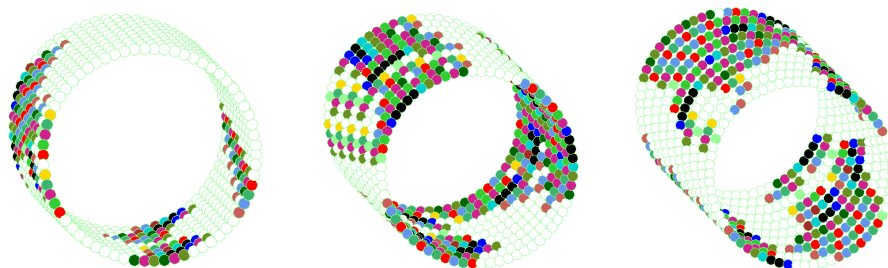


Figure 32.17: For diagonally scrolling network-graph space-time patterns, as in figure 32.15, key hits `e/c` expand/contract the spacing between time-steps by one pixel. These examples each have the same number of time-steps, but with expanded spacing — shown in 1d circle layout,  $n=66$ , 10 time-steps.

---

## 32.10 1d 2d 3d

Space-time patterns (STP) can be displayed in a dimension other than the “native” dimension. A 1d native network can be displayed in 2d or 3d, a 2d native in 1d or 3d, and a 3d native in 1d or 2d. The STP dimension can be preset to something other than the native dimension in section 31.2.5, or changes can be toggled on-the-fly, described here.

In addition, 2d STP can be toggled to 2d+time, or scrolled diagonally — this and 3d STP have additional on-the-fly options. A 3d native network when toggled to 2d STP results a stack of 2d horizontal slices (figure 32.18) following the method in figure 21.3 *Left*.

### 32.10.1 T..tog 1d-2d-3d

*if 2d+time is not active (section 32.10.2)*

**T** is a 3-way toggle between 1d-2d-3d STP — the order changes to show which is current and which is next in the toggle sequence, i.e. if 2d is active the prompt is **2d-3d-1d**, if 3d is active the prompt is **3d-1d-2d**. Note that 1d and 2d STP can be scrolled (section 32.13.3).

On-the-fly methods for converting between STP dimensions are as follows:

1d to 2d:  $i = \text{the smallest factor } \geq \sqrt{n}, j = n/i$

1d or 2d to 3d:  $i = \text{the smallest factor } \geq \sqrt[3]{n}, j = \text{the smallest factor } \geq \sqrt{n/i}, h = n/(i \times j)$   
For 1d prime  $n$ , the 2d and 3d representation will be a long string length  $n$ .

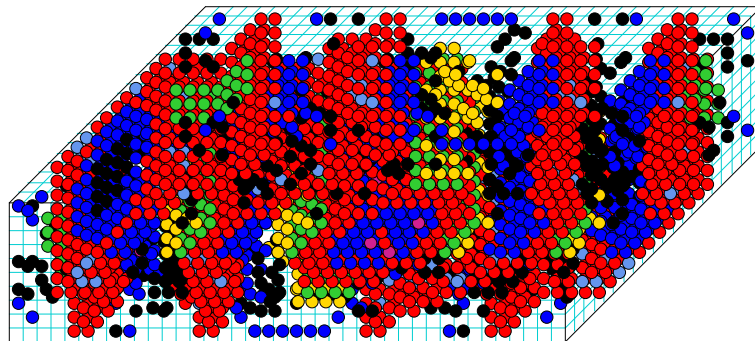
3d to 2d: The 2d representation consists of  $h$  horizontal slices (levels),  $i \times j$ , stacked one below the other starting from the top, i.e. a series of horizontal cross-sections through the 3d volume (figures 32.18, 21.3 *Left*).

2d or 3d to 1d: This will be represented as normal 1d space-time patterns, were the cells are indexed  $n - 1$  to 0 from left to right. The cell scale is automatically reduced to 1 pixel.



Figure 32.18: A 3d network shown in 2d by toggling 3d to 1d to 2d on-the-fly. *Below*: The 3d space-pattern,  $40 \times 20 \times 10$ . The projection is isometric seen from below, as if looking up into a cage.

*Left*: the 2d representation consists of 10 horizontal slices, one below the other, starting at the top, i.e. a series of horizontal cross-sections through the 3d volume. *v8k6* majority kcode but with shifted uniform outputs (section 16.8). The seed was a random central block which settled to this semi-stable pattern.



### 32.10.2 `t..tog 2d-2d+time`

*for 2d STP*

Enter `t` to toggle between 2d STP, and 2d plus a time dimension, drawn as a 3d isometric projection (figure 4.11). For binary ( $v=2$ ) networks, this is best seen when cells are colored according to the neighborhood (section 32.9.5). The current status is shown by the prompt sequence, i.e. if 2d+time is active the prompt is `t..2d+time-2d`. By default 2d+time will “sweep” – restart at the top on reaching the bottom of the screen. Enter `#` to scroll 2d+time (section 32.13.3).

### 32.10.3 `I..tog balls`

*if 3d STP, 2d+time, 2d diagonal scrolling*

If STP, 2d+time, or 2d diagonal scrolling is active, enter `I` on-the-fly to toggle between showing the normal current cell presentation and showing cells as balls (circles). or as flat parallelograms (figure 32.19) – also applies to 2d+time (section 32.10.2).

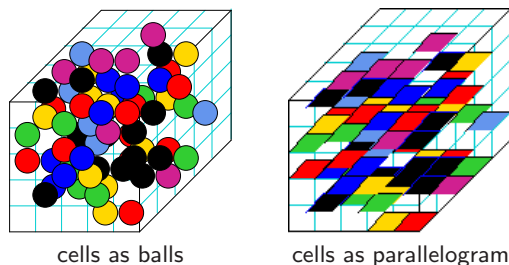


Figure 32.19: Toggling on-the-fly between showing 3d cells, *Left*: as balls or circles, or *Right*: as flat parallelograms.  $v=8$ ,  $n=5 \times 5 \times 5$ . Only the balls presentation can be captured as vector PostScript (section 21.4.11).

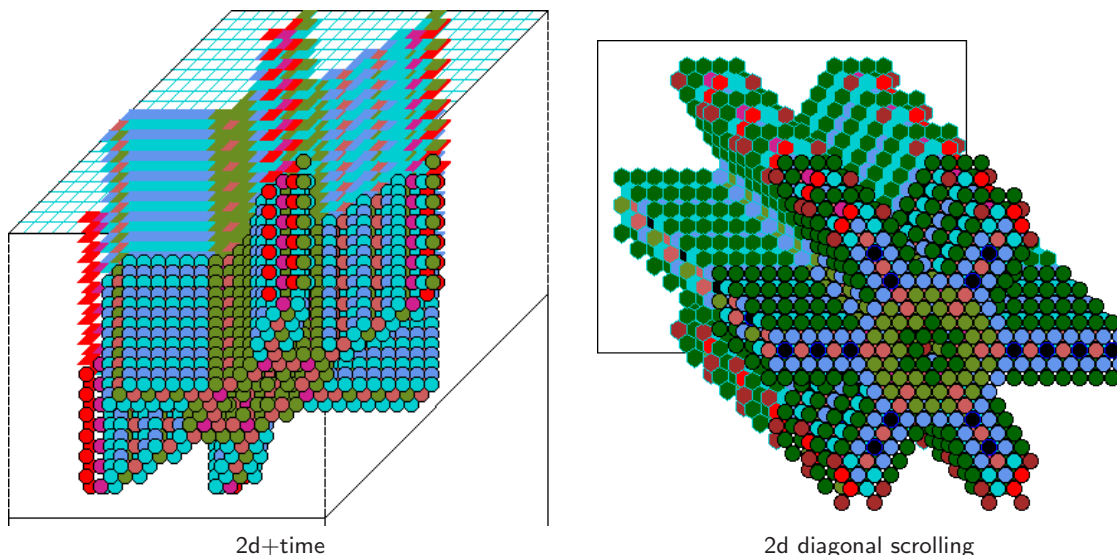


Figure 32.20: For 2d+time and 2d diagonal scrolling, key hit `I` toggles between the normal display and “balls”, here done at a halfway point in the time series. This example is for v3k6 kcode 19a08920520264,  $22 \times 22$  with the pattern having settled from a singleton seed.



### 32.10.4 p..plane

*for 2d+time or diagonal scrolling*

Enter **p** on-the-fly to draw a plane grid within the space-time patterns at the current time-step to highlight the geometry. This plane is also drawn whenever the seed, rule/s or wiring are changed. Figures 4.11 and 4.13 give examples.

### 32.10.5 J..invisible

*if 2d diagonal scrolling is active (section 32.13.3.2)*

Enter **J** on-the-fly to toggle (make invisible) normal diagonally scrolling space-time patterns. The purpose of this option is to see just the scrolling network-graph, if set (section 32.13.3.3 and figure 32.15).

## 32.11 Frozen/Filter

**filter** — *if input-entropy is active (section 31.5)*

“Frozen” options allow visualizing the activity/stability of cells. “Filter” options allow progressively filtering categories of cells according to neighborhood frequency (starting with the highest) to show up interacting configurations (gliders) more clearly. Note the frozen and filter options can be used together, and work for 1d, 2d and 3d (including RBN and DDN), but filtering 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 31.5.

### 32.11.1 h..nor-dy-f1-f2-bin

Enter **h** on-the-fly to toggle/cycle through the following five ways (figure 32.23 b,c,d,e,f) of displaying cells in space-time patterns: **nor** — the normal display — cells by value, **dy** — dynamic trace, showing dynamic time-trails (figure 32.21) — the most effective way of showing the path of gliders in 2d, **f1** and **f2** — two ways of showing “frozen” cells, and **bin** — frequency bins, then back to **nor**. Gliders time-trails also appear in **f1**, **f2** and **bin**, for example figure 4.12.

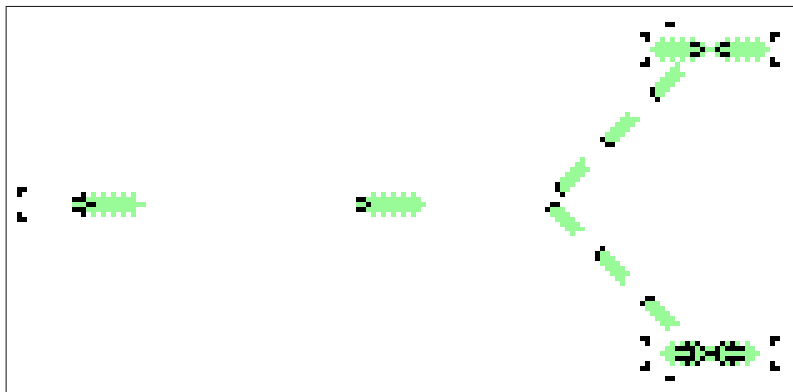
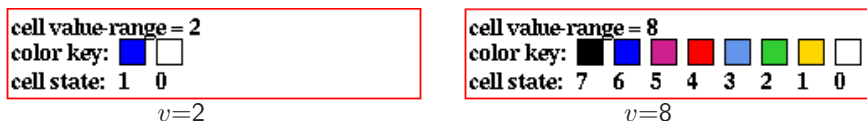
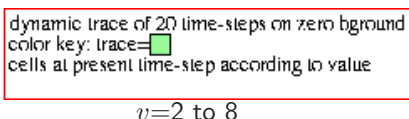


Figure 32.21: Glider-guns shooting interacting gliders, from the X-rule[13]. The green dynamic time-trails of 20 time-steps are activated with on-the-fly toggle/cycle key **h** (**dy** — dynamic trace, section 32.11). Enter **H** on-the-fly to change the length of time-trails (section 32.11.3).

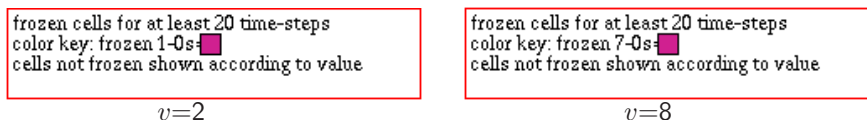
**nor** ... “normal” colors. The 5-way toggle at **nor** shows cells according to their actual cell value, even if color by neighborhood “lookup” was previously active.



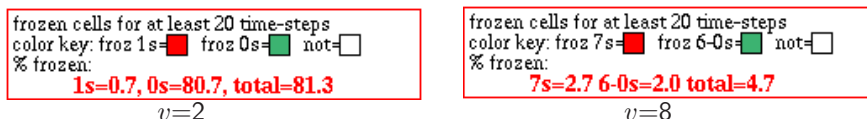
**dy** ... “dynamic trace” on a zero background, a new option, where gliders leave dynamic green trails, and the present moment is shown as “normal”. (figure 32.23 c).



**f1** ... “frozen” (method 1): frozen cells are colored purple. Dynamic, unfrozen cells, are show as “normal” (figure 32.23 c). The following color key reminder is shown in a top-right window,



**f2** ... “frozen” (method 2): for binary,  $v=2$ : frozen 1s are colored red, frozen 0s green. For  $v \geq 3$ : frozen  $(v-1)$ 's are colored red, frozen  $(v-2$  to  $0)$ 's green. Dynamic, unfrozen, cells are white (figure 32.23 d). The color key reminder also shows an updating percentage of reds, greens, and the total frozen.



**bin** ... colors are assigned from a spectrum representing the fraction of time each cell have been “on” in a (frozen generation) time window of  $x$  time-steps, so falling into a preset number of frequency bins, and bin boundaries (figure 32.23 e). For binary,  $v=2$ , “on” signifies 1. For  $v \geq 3$ , “on” signifies the maximum value  $(v-1)$ . The top-right reminder shows the bin color scheme, and bin boundaries defined by their upper limits (these parameters can be reset in section 32.11.2,

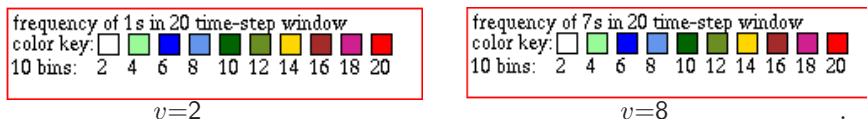


Figure 32.22: Top-right information and color keys for the 5-way toggle/cycle with on-the-fly key **h** for alternative presentations of space-time patterns .

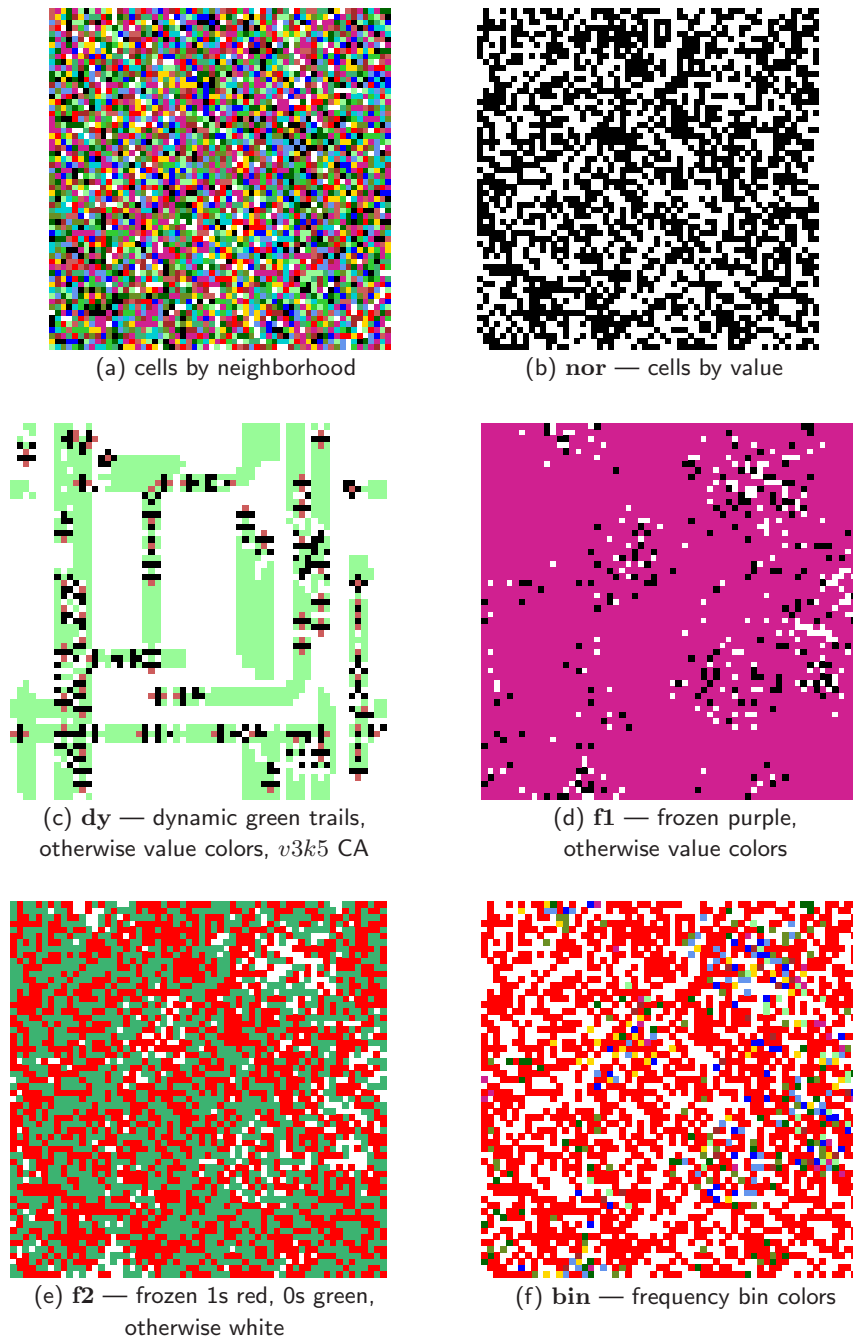


Figure 32.23: Toggling between (a,b) with `key 3` — cells by value or neighborhood (section 32.9.5), and 5-way toggling/cycling between (b/c/d/e/f) with `key h` — cells by value, dynamic traces (*v3k5* CA), two different “frozen” presentations, and frequency bins (section 32.11.1). All examples are on a 2D  $60 \times 60$  lattice. (c) is a new “dynamic trail” option where gliders leave green trails (*v3k5* CA (hex) 004a8a2a8254). (a,b,d,e,f) are the same *v2k5* RBN showing 5 alternative snapshots of the same stabilized dynamics with wiring restricted to an 8 cell radius, and mixed rules with canalizing inputs set to 51%.

The current status is given by the prompt order in the on-the-fly key index, for example **7.dy-f1-f2-bin-nor** indicates that the **dy** presentation is current. At the same time a top-right window gives the relevant color key and other information, figure 32.22. To qualify as “frozen” a cell’s value must remain unchanged according to parameters in section 32.11.3. Frequency bins are defined in section 32.11.4. Figure 32.23 gives space-time pattern examples.

### 32.11.2 H..f-gens (now 20)/bins(10)

Enter **H** on-the-fly to change “frozen” parameters, the default number of frozen generations, or the number of bins and bin boundaries, related to section 32.11.1 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 20, now 10):**

### 32.11.3 frozen generations

The “frozen” generation size is the number of time-steps that a cell must remain unchanged to qualify as “frozen”, and also sets the length of dynamic-trails (figure 32.21). It is reset with the following prompt,

**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 the on-the-fly index and becomes the new default.

### 32.11.4 frequency bins

To change the number of bins, and the bin boundaries, the following top-right prompt is presented after the frozen generation prompt in section 32.11.3,

**reset bin boundaries, enter new total (max 20, 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 (otherwise it is reduced automatically) – a subsequent top-right prompt sets 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 (example in figure 32.25). A frequency bin is the interval above the previous bin and its own upper limit. To define the bins, enter successive upper limits — each entry should be greater than the last. The following top-right prompts are presented in sequence,

**enter bin boundary 1 (def 2):**

**enter bin boundary 2 (def 4):**

*etc... invalid entries are corrected, by may also result in the following top-right warning*

**invalid bin boundaries, cont-ret:** *(enter return to backtrack)*

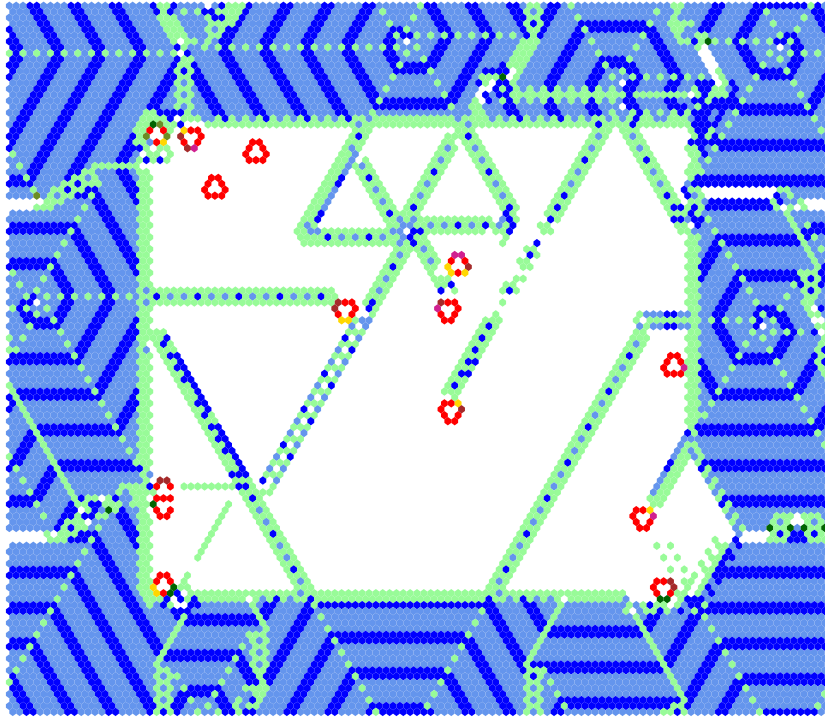


Figure 32.24: Space-time snapshot of a 2d CA inside another 2d CA with different  $k$ , the same space-time pattern as figure 19.6 (where rule details are given) but colored according to (default) frequency bins (section 32.11.4). The 2d ( $80 \times 80$ )  $v3k7$  CA spiral-rule[45] was loaded into a 2d ( $120 \times 120$ )  $v3k6$  complex CA which generates spiral structures.

frequency of 1s in 100 time-step window  
 color key:       
 5 bins: 1 5 20 40 100

Figure 32.25: Frequency bins set by hand shown — the top-right bin window (see figure 32.11.1). Frozen generation size = 100, no of bins = 5, bin boundaries: 1, 5, 20, 100.

### 32.11.5 f/F/a..filter/undo/all

*if input-entropy is active (section 31.5)*

On-the-fly filtering algorithms take streaming data from the input-frequency histogram (section 31.5), then suppress the printing of cells of the most frequent (unsuppressed) neighborhood. This suppresses the background domains first, being the most frequent. Gliders and other rare structures are left until last, so can be isolated and seen more clearly (figures 4.8, 32.26 — 32.28). The method works for any type of network (1d, 2d or 3d) provided that the input-entropy is active.

Filtering is implemented on-the-fly with the following key hits,

**f** ... to progressively filter space-time patterns.

**F** ... to unfilter in reverse order.

**a** ... to immediately restore the unfiltered pattern.

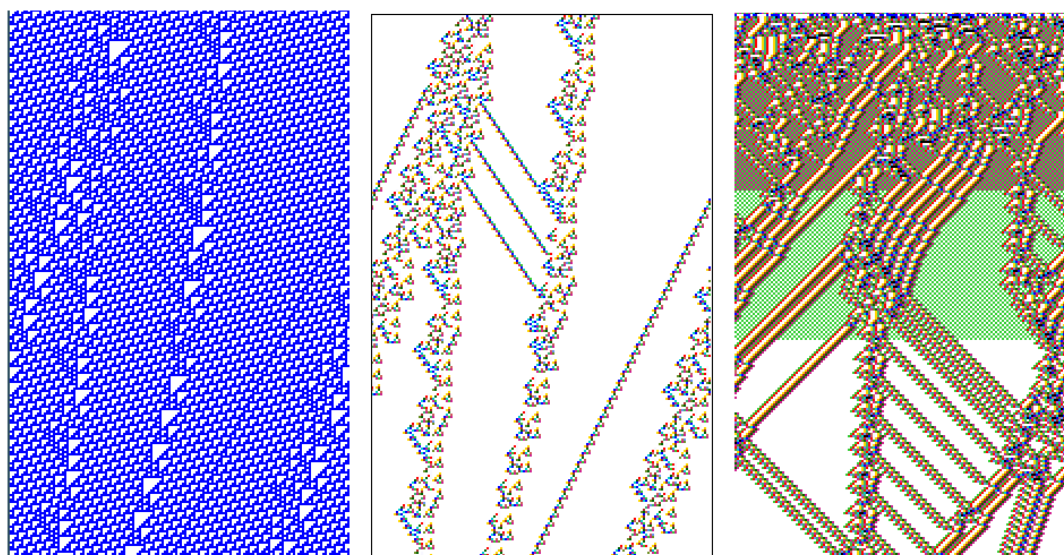


Figure 32.26: Examples of filtering space-time patterns to show up gliders more clearly ( $n=150$ , about 200 time-steps). *Left*: unfiltered space-time patterns of  $v2k3$  rcode 110, transformed to the equivalent  $v2k5$  rcode 3cfc3cfc,  $n=150$  (cells by value). *Centre*: the same space-time patterns — filtered (cells by neighborhood lookup). *Right*: space-time patterns of the  $v2k5$  rcode 360a96f9. Cells are shown by neighborhood lookup, and are progressively filtered in 2 stages.

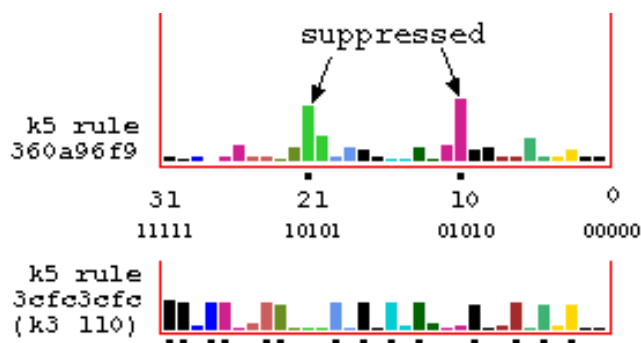


Figure 32.27: The lookup-frequency histogram relating to figure 32.26.

*Left Above*:  $v2k5$  rcode(hex)360a96f9.

*Left Below*:  $v2k3$  rcode(dec)110 transformed to  $v2k5$  rcode(hex)3cfc3cfc.

Suppressed neighborhoods are indicated with a dot. Note that the lookup frequency histogram is shown vertically in DDLab.

A dot is shown alongside the lookup-frequency histogram indicating which neighborhoods are currently suppressed (figure 32.27). As well as (and in addition to) progressively filtering/unfiltering on-the-fly, specific selected neighborhoods can be filtered in isolation (section 32.16.7).

For most glider rules, only a few neighborhoods need to be suppressed to filter domains. Rules with very complicated background domains, such as the  $v2k3$  rcode(dec) 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  $k=5$  neighborhoods (figures 32.28).

The method can also filter chaotic domains, and reveal discontinuities within or between chaotic domains (figure 32.29).

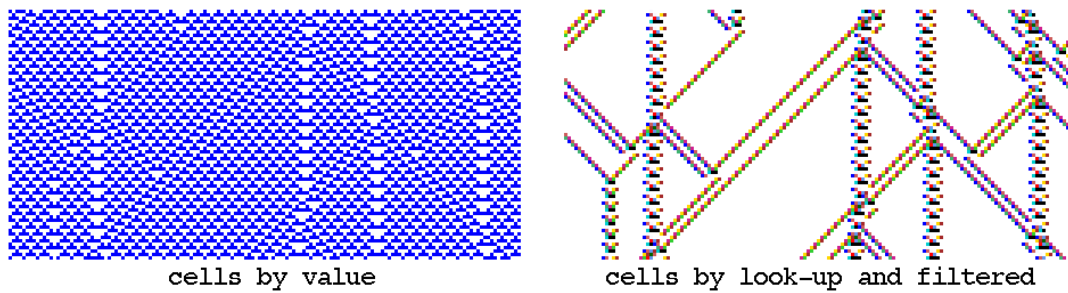


Figure 32.28: Space-time patterns from the same initial state showing interacting gliders, which are embedded in a complicated background. *Left*: unfiltered, cells by value. *Right*: cells by neighborhood lookup, with the background domain filtered. The  $k=3$  rule 54 was transformed to its equivalent  $v2k5=5$  rcode(hex) 0f3c0f3c,  $n=150$ .

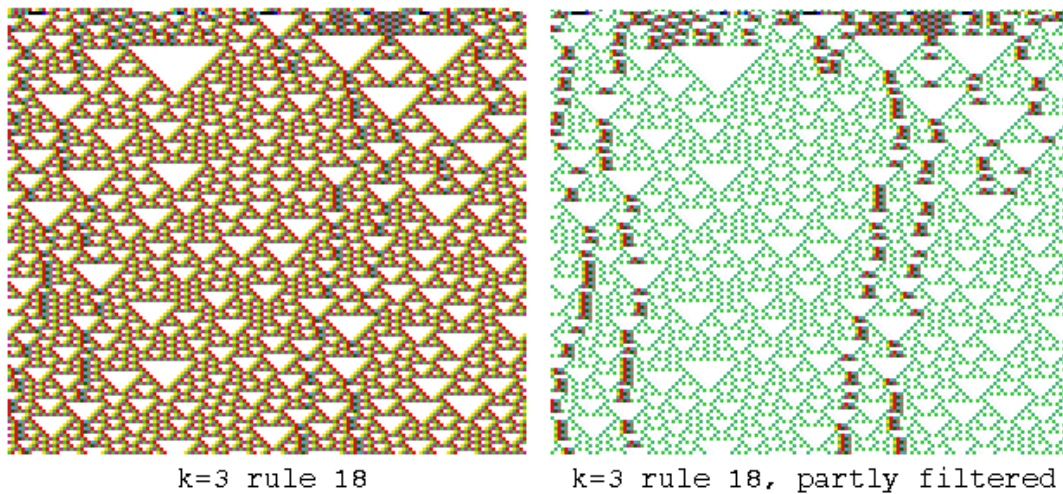


Figure 32.29: Unfiltered and partly filtered space-time patterns of  $v2k3$  rule 18, transformed to  $v2k5$  rcode(hex) 030c030c.  $n=150$ , about 130 time-steps from the same random initial state, showing discontinuities within the chaotic domain.

---

## 32.12 Analysis

These on-the-fly options relate to various methods of real-time analysis of space-time patterns, including the input-frequency histogram, input-entropy, pattern density, the entropy-density plot, return map, state-space matrix, and alternative presentations of the above.



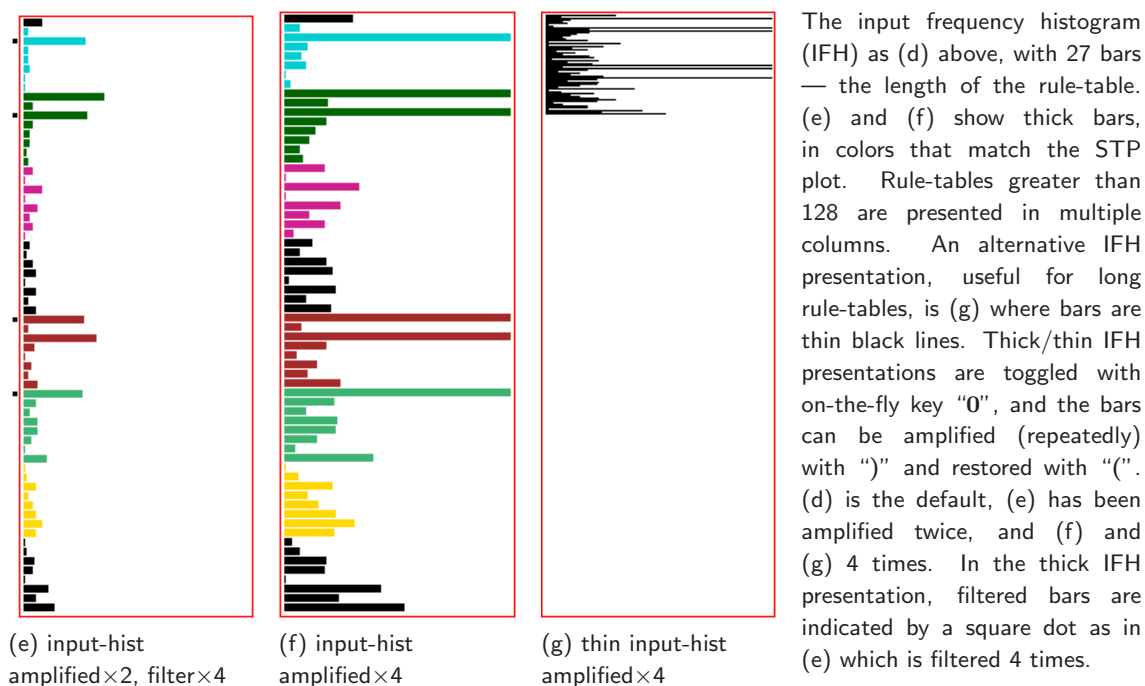
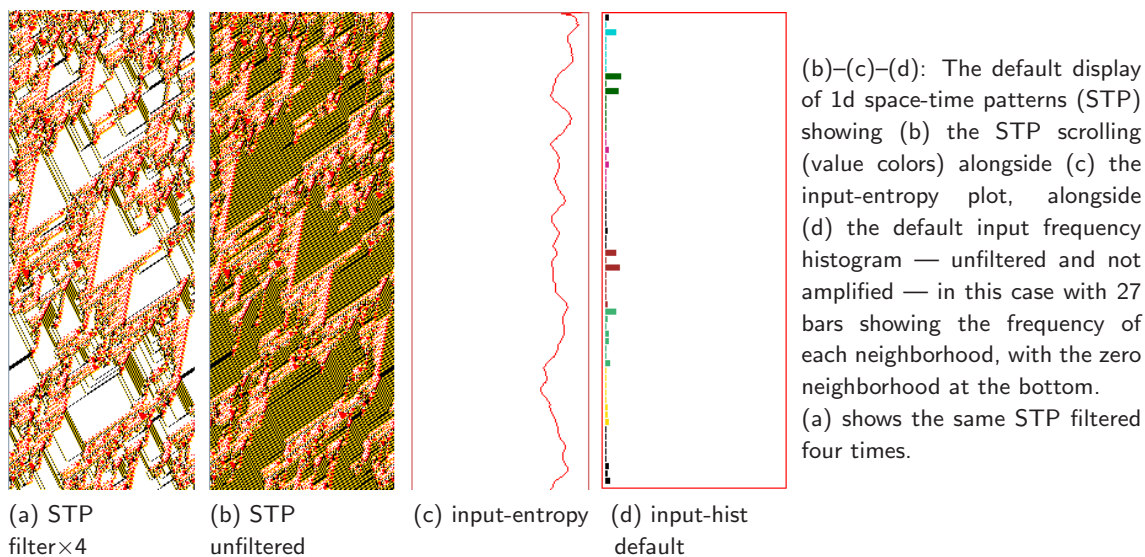


Figure 32.30: Input frequency histogram (IFH) presentation while running space-time patterns, showing the amplification of bars and which bars are filtered in space-time patterns. *v4k3* 1d CA, rcode (hex)f519751e8ea60489ff3ccaec94849968,  $n=150$ . 588 time-steps from the same random initial state.



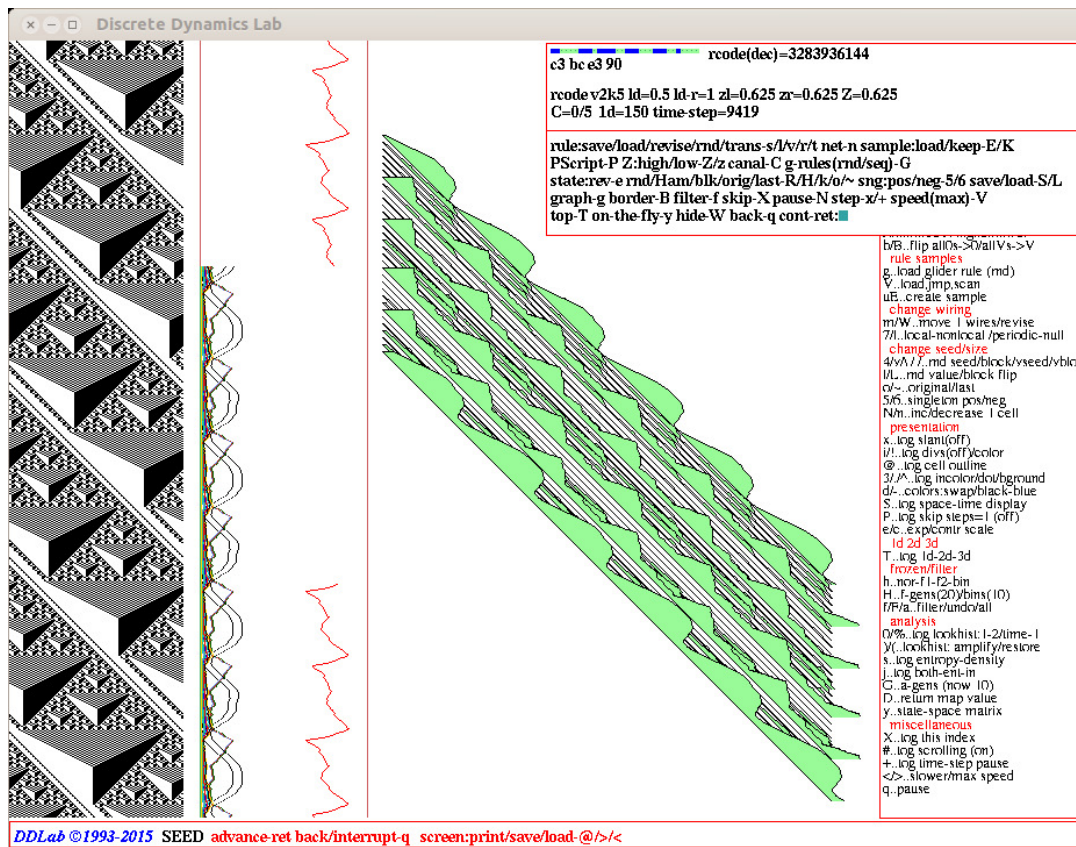


Figure 32.31: A snapshot of the DDLab screen, showing the input frequency histogram in 3d with an added time dimension. Various sections of the screen show the following: *Far Left*: a scrolling space-time pattern of a rule by Wentian Li, [23]  $v2k5$  rcode(hex)  $c3bce390$ ,  $n=150$ . *Near Left*: three alternative scrolling graphs (toggled with  $j$ , section 32.12.4) — input-entropy — the lookup frequency of each neighborhood — both simultaneously. *Center*: the input-frequency histogram in 3d (section 32.12.1). *Top Right*: the interrupt/pause prompt (section 32.16). *Far Right*: on-the-fly key index (section 32.1).

### 32.12.1 0/%..tog lookuphist:1-2/1-time

*if input-entropy is active (section 31.5)*

On-the-fly key hits to change the presentation of the input-frequency histogram are as follows,

- 0 ... (zero key) to toggle between normal and 1 pixel width bars (figure 32.30 g).
- % ... to toggle between the usual 2d histogram, expanded to 3d with a diagonal time axis — if the number of bars  $\leq 64$  (figure 32.31)

### 32.12.2 )/(..lookhist: amplify/restore

*if input-entropy is active (section 31.5)*

Enter on-the-fly key “)” to double the length of bars (can be repeated) — enter “(” to restore the default length (figure 32.30 d,e,f).

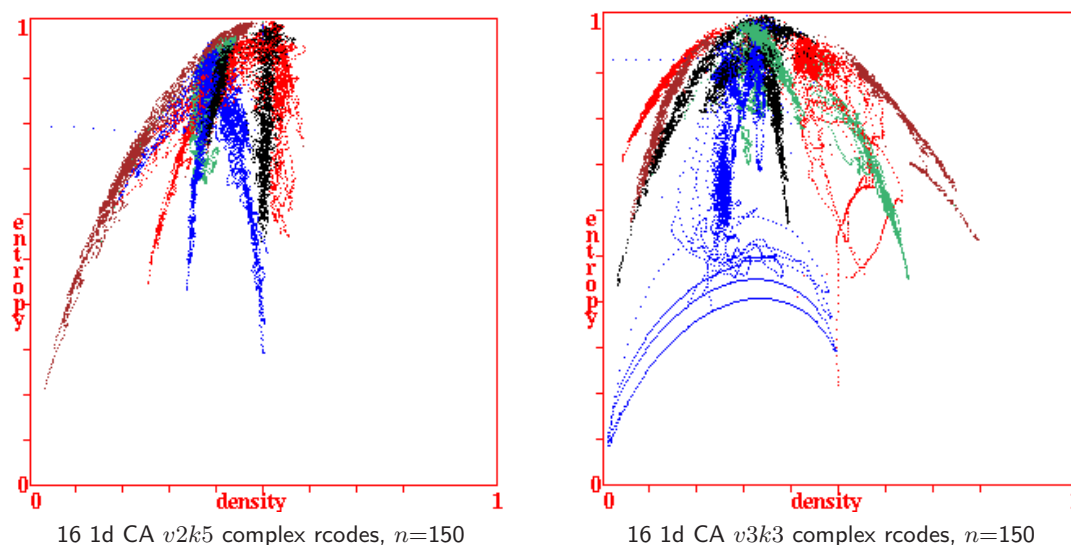


Figure 32.32: Entropy/density scatter plot[38]. The density is defined as follows: for binary,  $v=2$ , the fraction of 1s; for  $v \geq 3$ , the fraction of non-zero values. Input-entropy is plotted against the density relative to a moving window of time-steps, which can be reset in section 32.12.6. Plots for the first 16 rcodes in the complex rule collections (section 32.6.1) are show superimposed — each has its own distinctive signature, with high input-entropy variability. For each rule, about 1000 time-steps are plotted from several random initial states.

### 32.12.3 `s..tog` entropy-density

*if either input-entropy or pattern density is active (section 31.5)*

Enter `s` on-the-fly to toggle between showing the input-entropy, or the pattern densities as a set of  $v$  value-density graphs (section 31.5 and figure 31.8). The current status is given by the prompt order, i.e. **density-entropy** would indicates that **density** is current. These measures relate to a moving window of time-steps (default 10) that can be reset in section 32.12.6. For 1d networks the input-entropy and pattern density is the default, but 2d and 3d networks it needs to be set in section 31.8 to be active.

### 32.12.4 `j..tog` ent-in-both

*if input-entropy is active (section 31.5)*

On-the-fly key hit `j` is a 3-way toggle to show the input-entropy, or the input-frequencies as a set of graphs, or both together (figures 31.8, 32.31). The current status is given by the prompt order,

**ent-in-both** ... just the input-entropy graph.

**in-both-entropy** ... the input-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 31.5.

**both-ent-in** ... showing both of the above simultaneously.

### 32.12.5 **u..tog entropy/density plot**

*if input-entropy is active (section 31.5)*

Enter **u** on-the-fly to toggle showing the density/entropy scatter plot (figure 32.32) in a lower central window, where the entropy ( $y$ -axis) is plotted against the density ( $x$ -axis), relative to a moving window of time-steps (default 10), which can be reset in section 32.12.6).

Complex rules have distinctive scatter signatures, with high input-entropy variability, which lie within a parabolic envelope because high entropy is most probable at medium density, low entropy at either low or high density. Chaotic rules, on the other hand, give a compact scatter 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.

### 32.12.6 **G..a-gens (now 10)**

*if either input-entropy or pattern density is active (section 31.5)*

Enter **G** on-the-fly to change the default number of analysis generations, the size of the moving window of time-steps relating to input-entropy and pattern density (sections 32.12.3, 32.12.5).

The following top-right prompt is presented,

**enter new 'analysis' Generation size (now 10):**

The current analysis generations is shown in both the on-the-fly window and in this prompt. The initial default is 10 time-steps.

### 32.12.7 **D..return map by value**

Enter **D** on-the-fly to toggle a scatter plot of the return map by value (in a 2d-phase plane), described in section 31.2.2.2 and figure 31.5, providing characteristic fractal signatures for chaotic rules.

### 32.12.8 **;.return map by density**

If the space-time pattern density (section 32.12.3) is active, enter **;** (the semi-colon key) on-the-fly to toggle a scatter plot of the return map by density, plotting the density of each value at  $t_{-1}$   $y$ -axis, against  $t_0$   $x$ -axis, as in figure 32.33. This can be interesting when the densities follow a periodic oscillation, which occurs for a network with  $v \geq 3$  and random wiring but one complex rule, for example rules from the glider rule collections<sup>5</sup> selected on-the-fly with key **g** (section 32.6.1).

The best result are obtained if the generation size of the the moving window of time-steps (section 32.12.6) is reduced to one. The periodic density oscillations can be seen by the naked eye in the pulsating space-time pattern itself, but if the random wiring is localised (section 12.5.2), pulsating ceases but density waves reminiscent of reaction-diffusion are observed. To speed up the plot considerably, toggle the space-time pattern display off (on-the-fly **S**, section 32.9.8).

---

<sup>5</sup>To see the glider dynamics, toggle between CA and random wiring with key **7** (section 32.7.1.1).

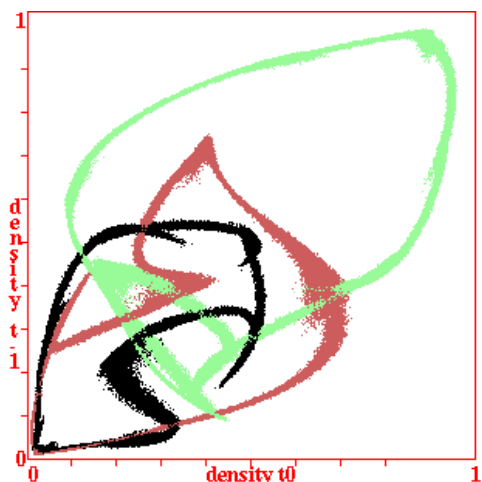


Figure 32.33: The return map by density, 2d hexagonal 222x222,  $v3k6$ ,  $kcode=(hex) 020282815a0254$ , with fully random wiring. The rule is from the complex rule collection `g_v3k6.r_v` (section 3.6.2) — similar results can be obtained with other rules, selected on-the-fly with key `g` (section 32.6.1). Colors green, red, black, represent values 0, 1, 2.

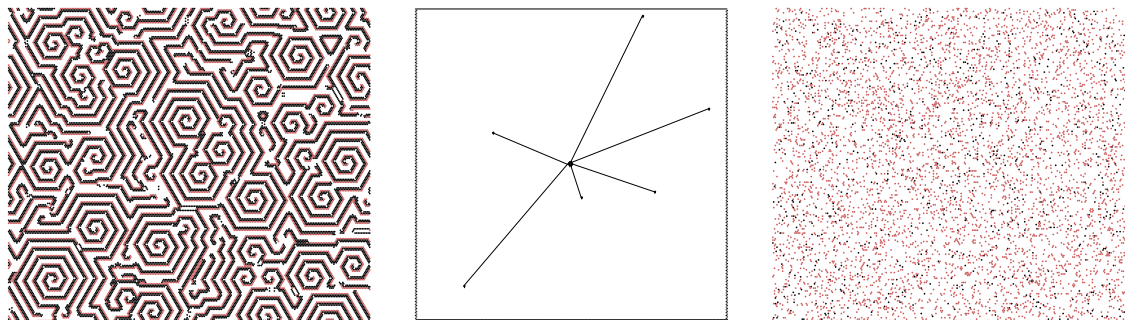


Figure 32.34: *Left*: A space-time snapshot of a 2d 222x222 cellular automaton with emergent spirals — the same complex rule as in (figure 32.33). *Center*: All connections were randomised (section 17.9.5) — the wiring of one central cell is shown. *Right*: A space-time snapshot of the resulting dynamics, which has immediately dissolved into chaos but with a pulsating density following periodic oscillation, captured in the density return-map (figure 32.33).

### 32.12.8.1 return map by density — saving the data

The return map density data can be saved to a file to enable analysis or visualisation in alternative software. To save the data, first pause space-time patterns (on-the-fly `q`), then enter `dnsty-d` (section 32.16). If saving the density is already in progress the following top-right prompt appears,

**now saving dnsty: stop-s, start new file-n, cont-ret:**

Otherwise, or if `n` is entered, a top-right filing prompt will appear (section 35.3) — the default filename is `my.dnsty.dat`. This is an ascii file — the first line gives the title “density return map:” and the start time-step, then an indefinite sequence (until interrupted) of densities for 0, 1, 2, . . .  $n-1$  where the values in each row must add up to one. For example, here are a few lines of data for from figure 32.33 where  $v=3$  so for densities of 0, 1, and 2,

```

density return map: start time-step=318
0.238171 0.263209 0.498620
0.442801 0.061460 0.495739
0.820956 0.009638 0.169406
0.953291 0.021873 0.024836
... continues

```

### 32.12.9 `y..state-space`

Enter `y` on-the-fly to toggle the “state-space matrix” which plots each state in the space-time pattern on a 2d grid plotting the left half of the bit string against the right half, described in section 24.5 and figure 31.4, providing a kind of signature of the dynamics.

### 32.12.10 `=..tog diff (keep damage)`

*if the “damage” is active (section 31.6)*

The method to graphically represent and analyse the difference, or the spread of “damage”, between the space-time patterns of two networks is described in section 31.6.

Enter “=” (the equals sign) on-the-fly to toggle between two definitions of damage, (**keep damage**) — once damaged keep the damage, and (**dontkeep damage**) — the instantaneous difference at each time-step. The on-the-fly index shows the current status.

## 32.13 Miscellaneous

These final miscellaneous on-the-fly options include pausing, scrolling, stepping, and changing the speed of space-time patterns.

### 32.13.1 `X..index display`

Enter `X` on-the-fly to toggle the on-the-fly key index itself.

### 32.13.2 `*..tog end pause (on)`

*for 1d or 2d+time space-time patterns*

Enter `*` (the star key) on-the-fly 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 is on, when the iterations reach the foot of the screen the interrupt/pause options (section 32.16) are presented in a top-right window. If the end pause is off, the sweeps continue indefinitely. The current status is shown in the prompt by **(on)** or **(off)**.

### 32.13.3 `#..tog scrolling`

*for 1d, 2d, or 2d+time space-time patterns*

Enter `#`, the hash key<sup>6</sup>, on-the-fly to toggle space-time pattern scrolling on/off, for 1d, 2d (including the network-graph), or 2d+time. These dimensions can be different from the network’s native dimension (section 32.10.1).

<sup>6</sup>The hash key, `#`, to toggle scrolling, may not give `#` on some keyboard settings (noticed in DOS) — in this case the “pounds” key, `£`, will probably give `#`, so try using `£` to toggle scrolling instead.

### 32.13.3.1 1d scrolling

The `#` key will toggle a 1d space-time presentation between scrolling upwards or making successive vertical sweeps (section 31.2.6). If set, the input-entropy or pattern density plots (sections 32.12.3 and 31.5) will also scroll or sweep.

### 32.13.3.2 2d diagonal scrolling

The `#` key will toggle a 2d space-time presentation between a 2d movie and scrolling diagonally upwards. When scrolling is set, successive 2d snapshots will move diagonally towards the bottom-right, then scroll diagonally upward, so that the present moment is the movie at the bottom-right (figure 4.13). An active network-graph will scroll simultaneously (section 32.13.3.3 below).

### 32.13.3.3 network-graph scrolling

If normal 2d space-time patterns (STP) are toggled (with the `#` key) to scroll diagonally upwards (section 32.13.3.2), an active network-graph (section 32.19) will also scroll simultaneously. Scrolling of both starts once normal 2d STP reaches the foot of the DDLab screen, with the present moment of both at the bottom-right. Enter `J` on-the-fly (section 32.10.5) to make the original 2d invisible and see just the scrolling network-graph (figures 4.9, 20.14).

### 32.13.3.4 2d+time scrolling

The `#` key will toggle a 2d+time presentation (section 32.10.2) between successive vertical sweeps and scrolling.

## 32.13.4 +..tog time-step pause

Enter `+` (the plus key) on-the-fly to pause after each time-step and other options (also available from the pause prompt, section 32.16.8). A top-right prompt is presented similar to the following,

```
time-step 6828, next-ret, reset count-r, save current-s: stepback x time-
steps (max=frozen-gens=19
on-the-fly/interrupt options=y/i, end pause-q: (values shown are examples)
```

Enter `return` for the next time-step. The prompt will reappear until *end pause-q* is entered. The full list of options is described below,

*options and info ... what they means*

**time-step 6828** ... the current time-step (for example).

**next-ret** ... enter `return` for the next time-step.

**reset count-r** ... to reset the count to 1.

**save current-s** ... to save the current space-time pattern.

**stepback x time-steps (max=frozen-gens=19** ... enter the number of time-steps to stepback and restart from a previous time-step. The maximum stepback is the frozen generation size, preset in section 31.2.3, or reset on-the-fly in section 32.11.2.





key to rule data

(the rule) ... its bit/value pattern (as much as will fit), in decimal (for small tables), and in hex (as much as will fit).

**rcode** ... or **kcode** or **tcode** — the rule type.

**v2k5** ... the value-range  $v$  and neighborhood size  $k$ . A  $k$ -mix will be indicated, for example **v2k(2-7)**.

for a full rule-table (rcode) only

**ld, ld-r** ...  $\lambda$  parameter[22],  $\lambda$  ratio[31].

**zl, zr, Z** ...  $Z_{left}$ ,  $Z_{right}$ , the  $Z$  parameter[31, 38].

**C=0/5** ... the number of canalizing inputs, in this case none, out a possible 5 ( $k=5$ ).

**\*3\*\*\*** ... shows exactly which of the  $k$  inputs are canalizing (if none this is not shown) — for example, 1 canalizing input, at neighborhood index 3.

For a mixed- $k$  network, rule date is omitted, so an example of the remaining data would be similar to the following,

```
rcode v2k(1-9) 1d-150 time-step=256
```

## 32.16 Space-time pattern interrupt/pause prompt

On pausing space-time patterns (on-the-fly **q**) the lower section of the top-right pause window<sup>9</sup> provides a range of context dependent interrupt/pause options<sup>10</sup> for amending/saving/loading the rule and state, and other functions including displaying the network, network-graph and generating a PostScript file, as well as any on-the-fly options from section 32.3.

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/+ speed-V dnsty-d
top-T on-the-fly-y hide-W back-q cont-ret:
```

Note that some of the options depend on the context, and only appear as follows,

options ... context

**trans-t** ... not in TFO-mode.

**Z:high/low-Z/z** ... single rule networks, and not in TFO-mode.

**canal-C** ... not in TFO-mode.

<sup>9</sup>The window can be temporally removed to uncover the hidden part of the screen with option *hide-W* (section 32.13.6).

<sup>10</sup>Some of these options are similar to “further attractor basin complete” options in section 30.5.



**g-rules(rnd/seq)-G** ... if a complex rule collection is detected for the current rule type,  $v$  and  $k$  (section 32.6.1).

**PScript-P** ... for 1d or 2d STP, irrespective of the native dimension. This includes 3d shown as 2d (sections 32.16.3, 32.18).

**graph-p** ... **p** appears if a network-graph is active — to save the current network-graph image to vector PostScript (section 32.19.2).

**dnsty-d** ... **d** appears if the “return map by density” (section 32.12.8) is active — to save the density data (section 32.12.8.1). .

Enter **q** to backtrack (initially to section 32.17). Enter **return** to continue space-time patterns, including any changes. The interrupt/pause options are described in the following categories,

	<i>category</i> ...	<i>section</i>
	revising rule/net ...	32.16.1
	classified samples of rule-space ...	32.16.2
capturing space-time patterns as a vector PostScript file ...		32.16.3
	revising the seed ...	32.16.4
space-time patterns on the network-graph ...		32.16.5
	fixed borders ...	32.16.6
	finer control of filtering ...	32.16.7
	skip, pause, step and speed ...	32.16.8
	miscellaneous options ...	32.16.9

### 32.16.1 Revising rule/net

The following rule and network options in section 32.16, are explained below,

**rule:save/load/revise/rnd/trans-s/l/v/r/t net-n** ... (**trans-t** not in TFO-mode)

**canal-C** ... (not in TFO-mode)

**Z:higher/lower-Z/z** ... (single rule networks only, and not in TFO-mode)

*options* ... *what they mean*

**save/load-s/l** ... enter **s** or **l** to save or load the rule (section 35.3). For mixed rule networks this applies to the rule at cell index 0 only.

**revise-v** ... to review/revise the rule in a lower-right window as described in chapter 16. 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)

**rnd-r** ... to reset a new random rule, or all rules at random in a rulemix.

**trans-t** ... (*rcode only*) to transform the rule as described in chapter 18. 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)

Note that the transform options allow canalizing inputs to be set for the whole network as well as for the selected rule (chapter 15).

- net-n** ... for the many network architecture options described in chapter 17, including viewing, revising, transforming, filing, rewiring, the Derrida plot (chapter 22), and learning (chapter 34).
- canal-C** ... (*not in TFO-mode*) to change or set canalizing inputs. For a single rule network the canalizing prompts comply with section 18.6 but are presented in a lower-right window. For a mixed rule network the canalizing prompts are described in chapter 15.
- g-rules(rnd/consec)-G** ... (*if a collections of complex rules is detected*) to choose either a random or consecutive rule from the collection (loaded on-the-fly with **g** — section 32.6.1). The current status is shown in the prompt order (i.e. **consec/rnd**). The same can be set in section 31.2.9.
- Z:higher/lower-Z/z** ... (*single rule networks only, and not in TFO-mode*) enter **Z** to progressively force the *Z*-parameter higher, or enter **z** to force it lower, as in section 32.5.5 and 16.3.

### 32.16.2 Classified samples of rule-space — load/keep

The following options for loading/displaying samples of automatically classified rule-space (chapter 33) are explained below.

**sample:load/keep-E/K**

*options ... what they mean*

**load-E** ... to load, then display, a new sample (section 33.5).

**keep-K** ... to keep, then display, a sample that is already loaded (section 33.5).

The samples can also be loaded on-the-fly (section 32.6.2), and scanned (section 32.6.3).

### 32.16.3 Directly scanning for PostScript

*option ... what it means*

**PScript-P** ... (*for 1d STP, or a snapshot of 2d or the 2d version of 3d*) enter **P** in section 32.16 to directly capture 1d space-time patterns (STP), or a snapshot of 2d STP, as a vector PostScript file, by directly scanning the DDLab image on the screen irrespective of the native dimensions — further detail in section 32.18. An advantage of direct scanning is that the PostScript colors will be the same as the screen image, according to the many alternative on-the-fly presentations.

For a native PostScript snapshot (1d, 2d or 3d) taken from memory see section 32.16.4 option **rev-e** or section 21.4.8.

### 32.16.4 Revising the seed and native PostScript

The following network state (seed) options in section 32.16, are explained below,

**state: rev-e rnd/Ham/blk/orig/last-R/H/k/o/~ sng:pos/neg-5/6 save/load-S/L**

*options ... what they mean*

**rev-e** ... to review/revise/save/load the state (seed) and many other options, which are presented in a lower center window (section 21.1). When **e** is entered, the following top-right prompt gives a choice of three possibilities, as follows,

**revise/reset seed: original-o last-l current-c:**

Enter **o** for the original seed, set or reset (here) in section 21.1. Enter **l** for the latest seed that was changed, either on-the-fly with any key hit in section 32.8, or in section 21.1. Enter **c** for the current state, at the time-step of the pause (section 32.16).

These options include vector PostScript capture (1d, 2d and 3d) according to native dimensions (sections 21.4.8, 21.4.11) — the only method method for the 3d isometric as in figure 32.19 *Left*.

**rnd-R** ... for a new random seed. The preset density-bias (section 21.3.2) of the seed (section 21.3) is respected — the same as on-the-fly key hit **4** (section 32.8.1).

**Ham-H** ... to change the values of a percentage of cells in the current state distributed at random. For binary networks ( $v=2$ ), the number of changed,  $H$ , is known as the Hamming distance. For  $v \geq 3$  the  $H$  cells will be assigned values at random that differ from the original values. The following top-right prompt is presented,

**enter % random Hamminig change to current state:**

**blk-k** ... for a central random central block of cells. The rest can be kept as is, or set to zero. The block size was originally specified in section 21.3 but can be changed here, and this will remain the block size for on-the-fly blocks set with **v** (section 32.8.1). The following top-right prompt is presented,

**rnd block: change size (now 30), rest: keep-k, all 0s-(def):**

If a new block size is entered the prompt reappears showing the new size. Enter **k** to keep the rest of the state, **return** for the new block on a zero background. The preset density-bias (section 21.3.2) block is are respected.

**orig-o** ... to restore the original seed set (or reset) in section 21.1.

**last-~** ... to restore the latest seed that was changed, either on-the-fly with any key hit in section 32.8, or in section 21.1.

**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 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** ... to enter the state (seed). When **S** is entered, the following top-right prompt gives a choice of three possibilities (as for *rev-e*) as follows,

**save seed: original-o last-l current-c:**

Enter **o** for the original seed, set or reset in section 21.1. Enter **l** for the latest seed that was changed, either on-the-fly with any key hit in section 32.8, or in section 21.1. Enter **c** for the current state, at the time-step of the pause (section 32.16).

Then save the state/seed — **.eed** file (section 35.3).

**load-L** ... to load a new seed — **.eed** file, (section 35.3).

### 32.16.5 Space-time patterns on the network-graph

**graph-g/p** ... (**p** if the network-graph is active)

Enter **g** to set up a network-graph (chapter 20), then run space-time patterns within it according to the graph layout and node sizes. This allows considerable presentation flexibility. Section 32.19 gives further details. The network-graph can be scrolled on-the-fly (section 32.13.3.3). Normal space-time patterns will continue to run simultaneously in the usual way unless disabled with on-the-fly (key hits **S**, or **J** if scrolling the network-graph). Enter **g** to deactivate graph space-time patterns if active.

Enter **p** to capture a snapshot of the current network-graph as a vector PostScript file (section 32.19.2 and chapter 36).

### 32.16.6 Fixed borders

**border-B** ... By default, networks in 1d, 2d and 3d have periodic boundaries, but this can be effectively eliminated if required, creating “falling off the edge” boundaries by fixing the value of border (edge) cells. This can be useful to absorb glider and other propagating structure which would otherwise “wrap around” and disturb the central space-time pattern.

If **B** in section 32.16, the following consecutive prompts are presented to set the border width and value, in a top-right window,

**fixed border: width(0-9, now 1):** (*for example*)  
**border value: (0-2), now 0):** (*for v=3*)

### 32.16.7 Finer control of filtering

*if input-entropy is active (section 31.5)*

**filter-f** ... Enter **f** in section 32.16 for finer control of filtering than in section 32.11.5. The following top-right prompt is presented (for example, for the  $v2k5$  CA in figures 32.26 and 32.27),

**filtered: 21 10** (*if unfiltered none shown here*)  
**filter/undo enter +/- k-index (31-0), max-m reset-r:**

**filtered: 21 10** show the neighborhood indexes that are currently filtered, otherwise **none** is shown. Specific neighborhoods can be filtered and unfiltered. For example, to filter neighborhood index 26 enter **+26**, to unfilter neighborhood 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, and filtered neighborhoods will be indicated by dots alongside the bars in the lookup-frequency histogram.

### 32.16.8 Space-time — skip, pause, step and speed

The following options to skip, pause, backtrack and step through time-steps, in section 32.16, are explained below, (see also section 31.2.7),

**skip-X pause-N step-x/+ speed(max)-V** (or **speed(slow)-V** if not *max*)

*options ... what they mean*

**skip-X** ... to change the number of time-steps to be skipped with on-the-fly with key hit **P** (section 32.9.9). The following top-right prompt is presented,

**skip time-steps (now 1):** (for example)

The default is (**now 1**) to skip 1 time-step, i.e. showing every second time-step, so if changed to (**now 2**), every third time-step will be shown, etc. To see the effect, toggle skipping steps to **P..tog skip steps=1 (on)** — shown in the on-the-fly key index (section 32.1). Figure 32.16 gives an example.

**pause-N** ... 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 always occurs when the screen is full, as in section 32.13.2 (takes effect only for 1d and 2d+time space-time if not scrolling), 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 32.11.2 (default 20), or enter a number for a one time pause after that number of time-steps.

**step-x** ... 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 with **pause-N** above (default 20).

**step-+...** to pause after each time-step and other options. A top-right prompt is presented similar to the following,

**time-step 66, next-ret, reset count-r, save current-s  
stepback x time-steps (max=frozen-gens=19  
on-the-fly/interrupt options=y/i, end pause-q:**

Enter **return** for the next time-step. The prompt will reappear until **end pause-q** is entered. This is also an on-the-fly prompt (section 32.13.4) where the full list of options is described.

**speed-V** ... the current speed status is either (**max**) or (**slow**). Enter **V** to reduce the speed of space-time pattern iteration, or to restore full speed, as in section 31.2.8, which also points to other stages in DDLab where slow motion can be invoked, including on-the-fly.

### 32.16.9 Miscellaneous options

*options ... what they mean*

**top-T** ... to continue 1d (and 2d+time) space-time patterns starting at the top of the screen. This also resets the time-step count.

**on-the-fly-y** ... to activate any on-the-fly option during a pause.  
The following top-right prompt is presented in a short window,

**enter on-the-fly option:**

Enter any key listed in the current on-the-fly key index (as in figure 32.1). The pause prompt (section 32.16) will reappear to make other changes if required — enter **return** to continue the space-time pattern, where the on-the-fly option selected will take effect.

**hide-W** ... to reveal what's hidden beneath the pause/interrupt window — enter **W** — any key will restore the window (see also section 32.13.6).

**back-q** ... to backtrack from space-time patterns, initially giving top-left options described in section 32.17.

**cont-ret** ... resume space-time patterns from before the pause, with changes taking effect.

## 32.17 Quit and further options

Enter **q** in section 32.16 to quit space-time patterns and backtrack. The following top-left prompt is first presented,

```
graphics-g speed(max)-V (or speed(slow)-V if not max)
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,

*options ... what they mean*

**graphics-g** ... to alter the graphics setup described in section 6.3.

**speed-V** ... the current speed status is either (**max**) or (**slow**). Enter **V** to reduce the speed of space-time pattern iteration, or to restore full speed, as in section 31.2.8, which also points to other stages in DDLab where slow motion can be invoked, including on-the-fly.

- seed-e** ... to revise the seed. The seed options described in chapter 21 are presented in a lower central window, a repeat of **rev-e** in section 32.16.4.
- 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), a repeat of **net-n** in section 32.16.1.

## 32.18 Directly scanning STP for vector PostScript

*1d STP, or a snapshot of 2d, or the 2d version of 3d*

The space-time image on the screen can be directly scanned to generate a vector PostScript (\*.ps) file (default filename `my_sps.ps`). This is the only method available for 1d space-time patterns (i.e. including time-steps), and one of the possible methods for a snapshot of 2d or the 2d version of 3d. An advantage of direct scanning as opposed to using information about a state held in memory<sup>11</sup> is that the PostScript colors will be exactly as presented at that moment, according to the many alternative on-the-fly presentations options, such as color by neighborhood, filtering, and frozen cells. For 1d there could be new seeds and rules within one space-time pattern (figure 32.35).

Direct scanning still allows resetting the scale, divisions between cells, dots on zero cells, and selecting greyscale or color, and follows the current screen dimensions (section 32.10) irrespective of the native dimensions.

Enter **P** in section 32.16 for the top-right PostScript options.

For 1d space-time patterns, direct scanning is the only method, and just part of the space-time pattern can be scanned — the prompt starts as follows,

```
create PostScript image for 1d space-time patterns, exit-q
1d size (max/def=150):    time-steps (max/def=669): (values shown are examples)
```

*options ... what they mean*

**1d size (max/def=150)** ... enter the space to include, the number of cells horizontally from the left, the maximum and default is what is visible.

**time-steps (max/def=669)** ... enter the number of time-steps from the top of the screen down, the maximum and default is what is visible.

For a 2d snapshot (figures 32.7, 32.23, 32.24), direct scanning (**direct-d**) is just one of several possible methods, the others are described in section 21.4.10, – the prompt starts as follows,

```
create PostScript image for 2d, 40x40 (values shown are examples)
symbols-s greyscale-g color-c direct-d exit-q (def-c):
```

Enter **d** for direct scanning.

For both 1d space-time patterns, and a 2d snapshot, the top-right prompts continue as follows,

```
greyscale-g color-c (def-c): (the default follows the last selection)
```

<sup>11</sup>For a PostScript snapshot of the *native* space-time pattern held in memory (1d, 2d or 3d), enter *state:rev-e* in section 32.16, select the state required (original, last, or current), then follow the seed methods in sections 21.4.8 and 21.4.11.

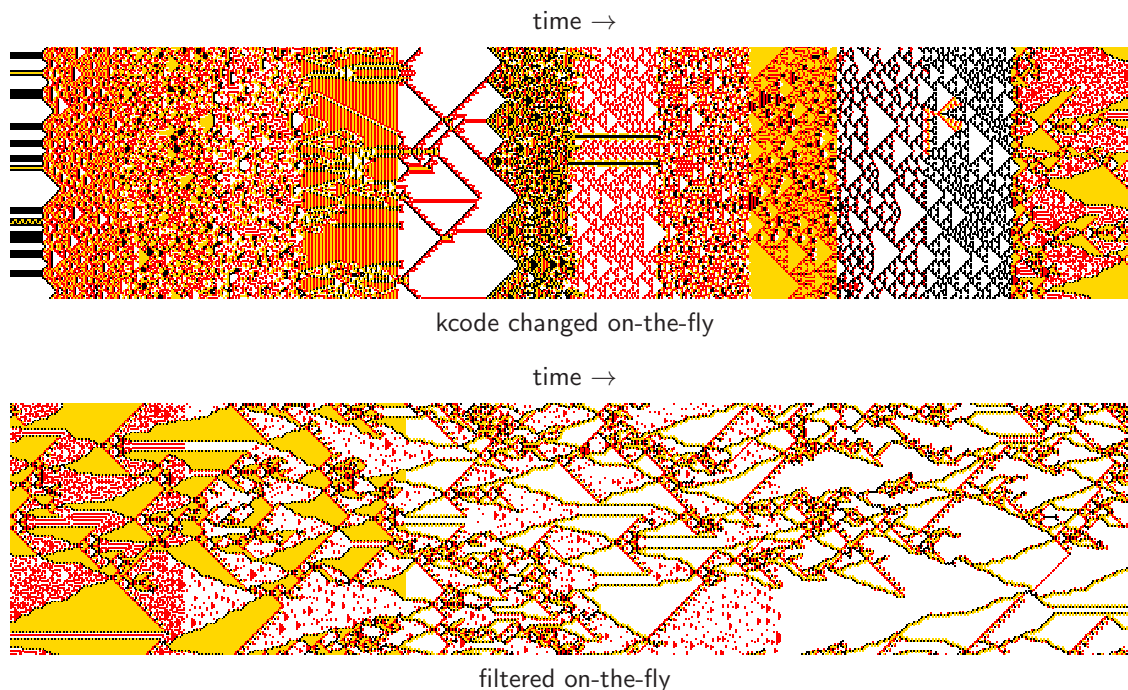


Figure 32.35: Directly scanned 1d space-time patterns for vector PostScript,  $v4k3$  kcode,  $n=150$ , 669 time-steps. The images shown are rotated by  $90^\circ$  so time flows from left to right.

*Top:* the rule was changed randomly on-the-fly (section 32.5.1) several times as the space-time pattern scrolled. Each new rule picked up the current state as its initial state.

*Bottom:* the space-time pattern was allowed to run on keeping the last rule,  $v4k3$  kcode (hex 89d17b2278), and the space-time pattern was filtered 4 times on-the-fly (section 32.11).

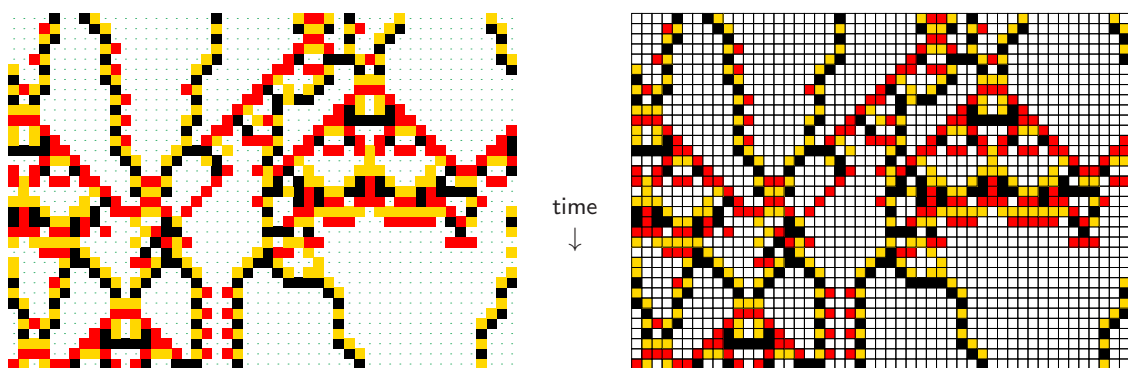


Figure 32.36: PostScript images from a scanned 1d space-time pattern, for the same  $v4k3$  kcode as in figure 32.35,  $n=50$ , 35 time-steps. *Left:* dots in zero cells. *Right:* black divisions between cells.



*options ... what they mean*

**greyscale-g:** ... for greyscale instead of color.

**color-c (def-c):** ... for color instead of greyscale.

This is followed by the prompt to amend other settings,

**cellscale=1.00 dots(off)=0.70 divs(off), 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. The options are summarized below,

*options ... what they mean*

**cellscale** ... enter a new cell width in pixels.

**togdots-x** ... to toggle zero dots on/off.

**dotscale** ... (*if dots are on*) enter a new width, which can be a decimal number, for the size of dots on cells with value zero.

**divs(0,1,2)** ... enter the cell division color, **1** for black, and **2** for white, or **0** for none — no division so adjoining cells touch. The new designation becomes the default. The current status is shown in the previous prompt line, **divs(1)**, **divs(2)**, or **divs(off)** for none.

Once these choices are complete, the \*.ps file is saved from the filing prompt (section 35.3). section 36.1 explains how to view, edit, and crop the PostScript image.

## 32.19 Network-graph layout of space-time patterns

Space-time patterns can be shown running within the network-graph layout (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 (square or hex), 3d, as well as allowing dragging nodes and components, and many other options for rearranging the graph.

Enter **g** in section 32.16. 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,

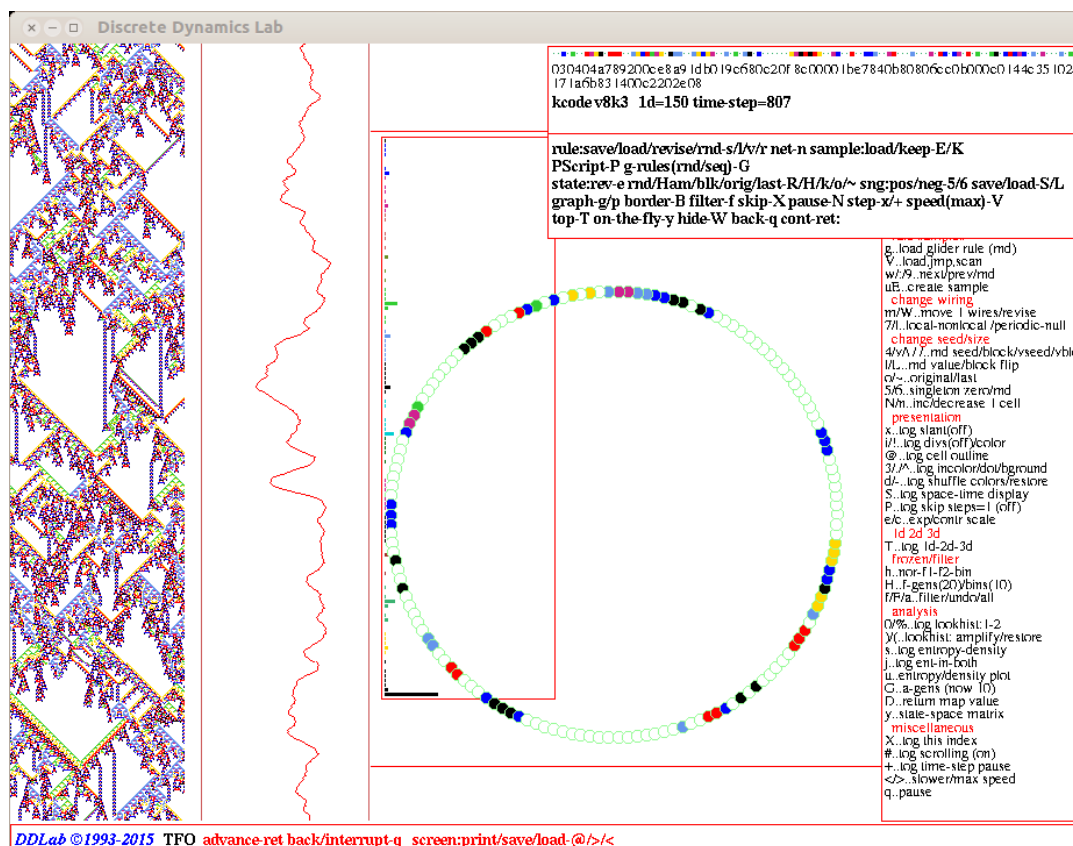


Figure 32.37: A snapshot of the DDLab screen, showing 1d CA space-time patterns (STP) according to the (default) circle network-graph layout, which reflects the actual geometry of a 1d CA with periodic boundary conditions.  $n=150$ ,  $v8k3$  kcode, index 8 from the rule sample in figure 33.5. The normal space-time patterns are on the left of the screen. Many on-the-fly options apply simultaneously to the normal and network-graph layout STP.

**NETWORK-graph (no links):** drag-(def) PScript-P tog:win-w  
settings-S rotate-x/X flip-h/v, exp/contr: nodes-e/c links-E/C both-B/b  
nodes-n/N  
layout: file-f circle/spiral-o/O 1d/2d(tog)/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),

**node 205, single:** drag/release - left mouse button, tog drag-d  
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 patterns will be shown according to the final network-graph layout in a large central window, simultaneously with the normal space-time patterns.

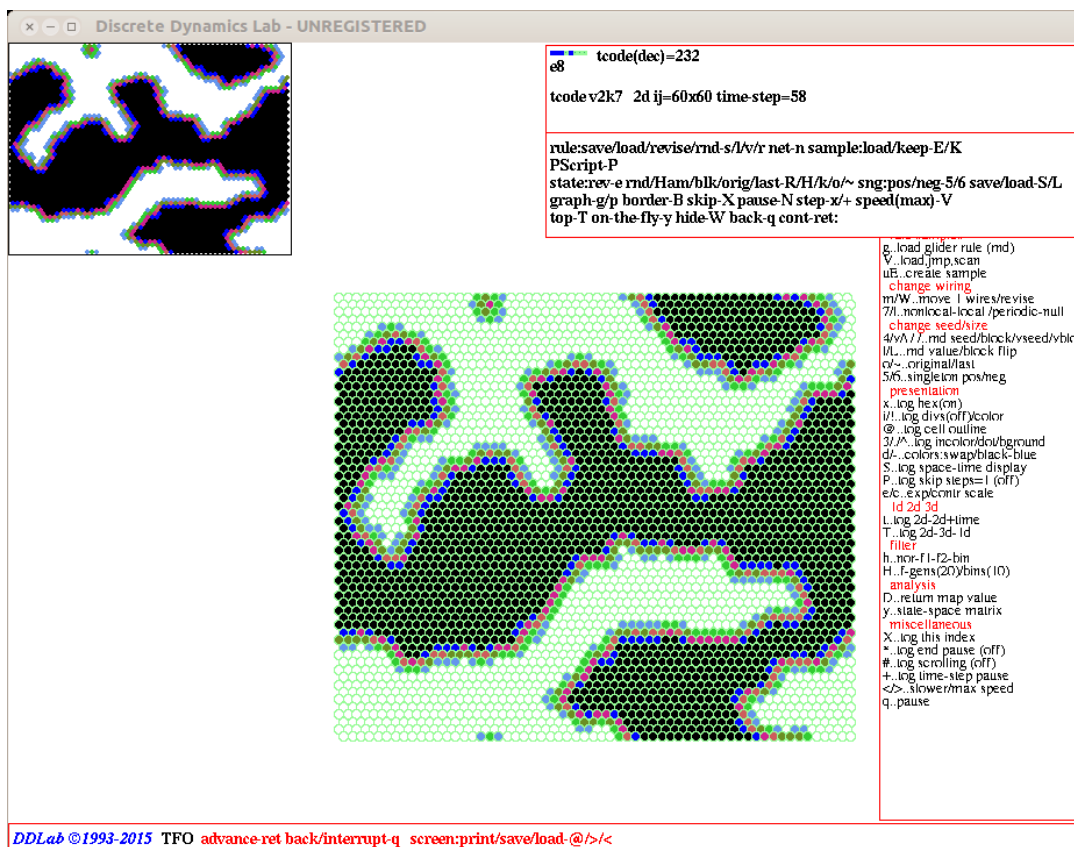


Figure 32.38: A snapshot of the DDLab screen, showing *Center*: 2d CA space-time patterns within the network-graph. *Top-Left*: the normal 2d space-time patterns continue at the top-left of the screen. Most on-the-fly options apply to both normal and network-graph space-time patterns.  $v2k7$  tcode(dec) 232 (a modified majority rule)  $n=60 \times 60$  with hexagonal layout and color by neighborhood.

### 32.19.1 On-the-fly options within the network-graph

Most current on-the-fly options (listed in figure 32.1) have the same simultaneous effect on both normal space-time patterns and space-time patterns within the network-graph, including colors and filtering. However some key hits have no effect or have special functions within the network-graph, as follows,

*on-the-fly category* ... *key hits with no effect, or special functions in the network-graph*

*change seed/size* ... **N/n.inc/decrease 1 cell** (only for 1d single-rule networks presented in 1d, section 32.8.6) — turns off the network-graph. Otherwise no effect.

*presentation* ... **x.tog slant(off)** or **x.tog hex(off)** (sections 32.9.1 or 32.9.2) — no effect.

**@.tog balls outline** (sections 32.9.4 and 32.19) — applies only to the network-graph.

**i/!..tog divs(off)/color** (section 32.9.3) — no effect.

**3/./^..incolor/dot/bground** (section 32.9.5) — **3** and **^** apply to the network-graph, but **.”** (dot) has no effect.

**S..tog space-time display** (section 32.9.8) — no effect.

**e/c..expand/contr scale** (section 32.9.11) — no effect unless diagonally scrolling the network-graph — then **e/c** expands/contracts the spacing between network-graph time-steps as in figure 32.17.

*1d 2d 3d* ... no effect by any on-the-fly key hits.

*analysis* ... no effect by any on-the-fly key hits.

*miscellaneous* ... **#..tog scrolling** (section 32.13.3) — if 2d STP is active, scrolls both 2d STP and the network-graph simultaneously (section 32.13.3.3) — otherwise no effect.

### 32.19.2 Network-graph space-time pattern as vector PostScript

If the network-graph space-time pattern is active, the current time-step can be saved to a vector PostScript file. Enter *graph-p* in the pause options (section 32.16). The following top-right prompt is displayed,

**save network-graph snapshot to PScript: 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 35.3) will be displayed to select the filename (default *my\_sgPS.ps*).

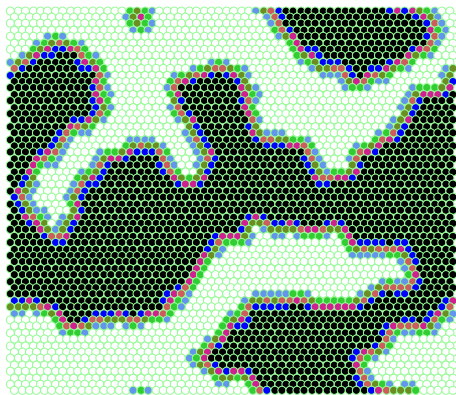


Figure 32.39: A vector PostScript image of network-graph in figure 32.38 — the same 2d CA space-time-step.

## Chapter 33

# Classifying rule space

This chapter describes methods to automatically classify samples of CA rule-space by input-entropy variability to distinguish between ordered, complex and chaotic rules (figure 2.8), to estimate their distributions in rule-space (e.g. figure 33.1), and to search for interesting rules. The methods include creating, sorting, plotting, analysing and probing the sample. Originally applied to 1d binary rcode [35, 38], the methods apply equally to 2d and 3d, to  $v \geq 3$ , and to kcode and tcode in TFO-mode [44]. Sample files used as examples in this chapter are provided with DDLab (section 3.6.4). The collections of complex or “glider” rules (sections 3.6.1, 32.6.1) were mostly picked out from these sample files.

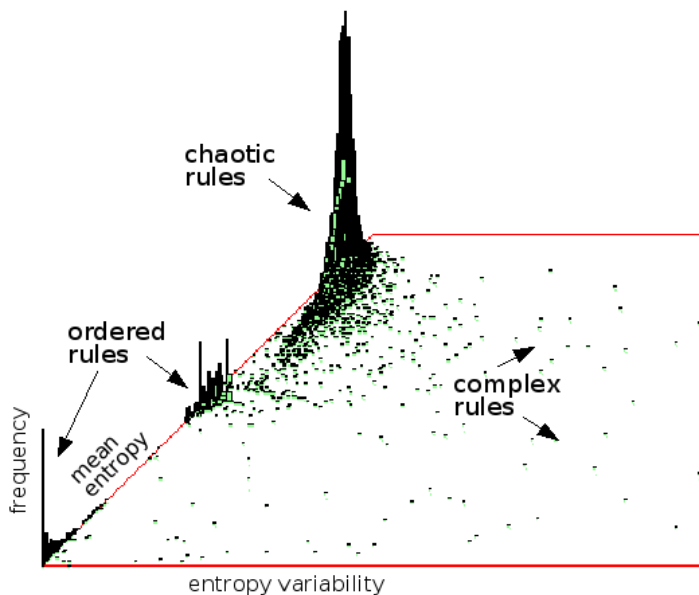


Figure 33.1: A 2d histogram for a sample of 15800 2d hexagonal unbiased CA kcodes *v3k6* (file *v3k6bs.sta*), plotting mean entropy against entropy variability (standard deviation). The method automatically classifies rules between ordered, complex and chaotic dynamics [43, 44]. The vertical axis shows the frequency of rules on a  $256 \times 256$  grid — most are chaotic.

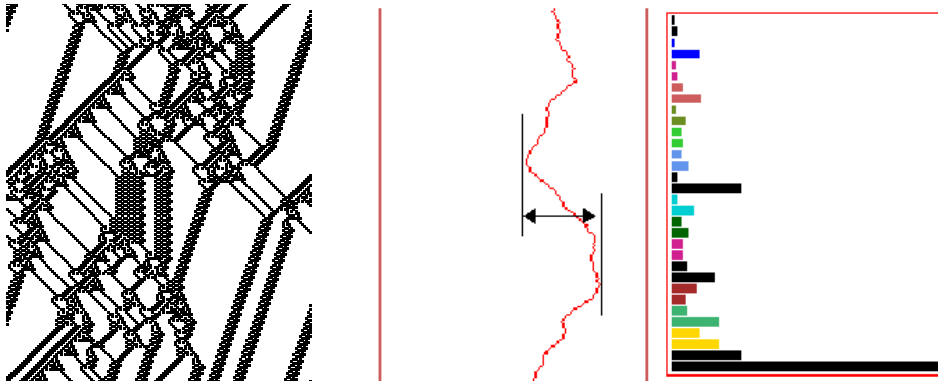


Figure 33.2: *Left*: the space-time patterns of a 1d complex CA with interacting particles,  $v2k5$  rcode (hex)6c1e53a8,  $n=150$  about 200 time-steps. *Right*: a snapshot of the input frequency histogram measured over a moving window of 10 time-steps. *Center*: the changing entropy of the histogram; its variability, either min-max (maximum up-slope as indicated) or standard deviation (sdev), provides a non-subjective measure to separate complexity from either order or chaos — high variability implies complex dynamics.

Rules can be generated at random or with some bias. Biases include isotropic rules, fixing the  $\lambda$  parameter, or fixing the proportions of different values in the rule-table. In addition, the rule-table can be adapted so that the all-zero neighborhood outputs zero, or more generally all- $V$  neighborhoods output  $V$ . These biases make complex dynamics more probable.

For each successive rule, the space-time pattern is run from a set of random initial states. For each initial state, after a delay to allow the CA to settle into its typical behavior, the variability of the input-entropy and the mean entropy are recorded — these measure are described in section 33.1. Then the average results from the set of initial states are plotted — the entropy variability ( $x$ -axis) against the mean entropy ( $y$ -axis), and the data for each rule is appended to a `.sta` file. The results of an automatic classification can be interpreted as follows,

	order	complexity	chaos
mean entropy	low	medium	high
entropy variability	low	high	low

High variability implies large scale structural interactions, often produced by particles colliding, because collisions create local chaos raising the entropy, from which particles re-emerge lowering entropy. The basic argument is that if the entropy continues to vary sufficiently in settled dynamics, moving both up and down, then some kind of self-organizing collective behavior must be unfolding. This might include competing zones of order and chaos, or different types of competing chaos, as well as glider dynamics.

So complexity can be separated by its high variability. Both order and chaos have low variability, but can be separated from each other by entropy (figure 33.1), though the probability of order diminishes with increased  $v$  and  $k$  in an unbiased sample. Once the sample is sorted, by variability followed by mean entropy, rule dynamics can be scanned by decreasing variability, from the highest at index 1 or any other starting point. The most interesting rules can be saved or added to collections of glider rules (section 32.6.1).

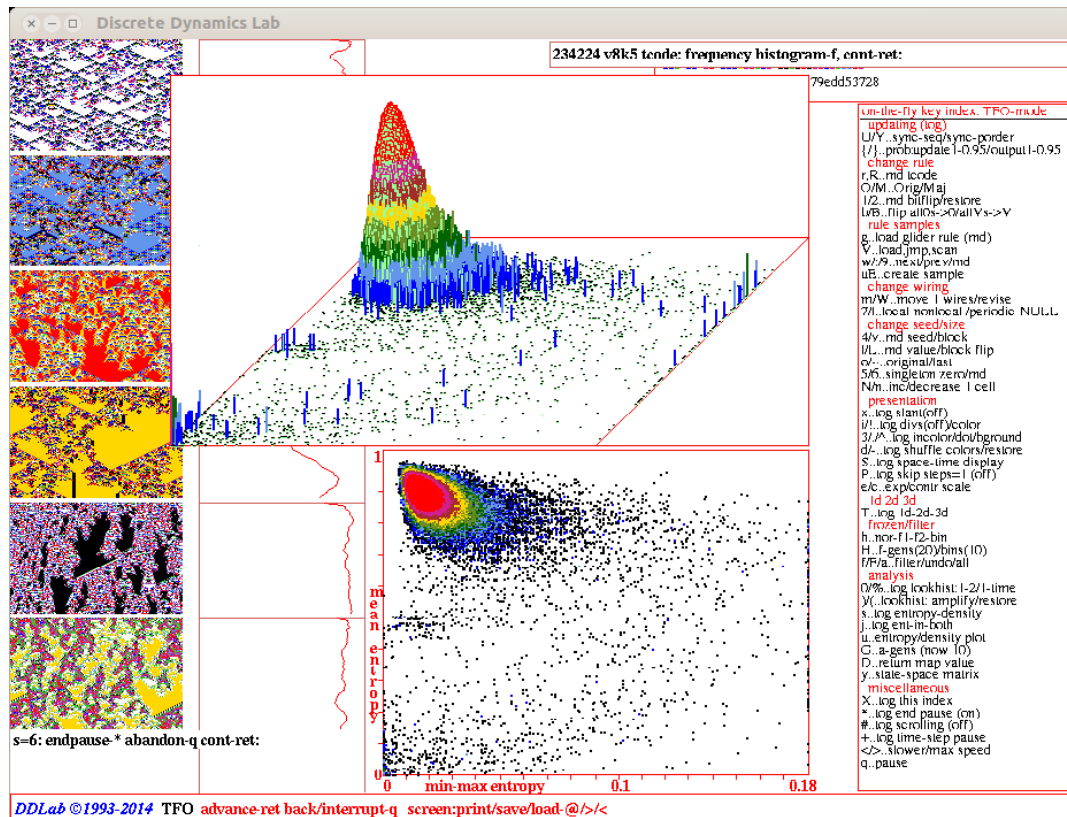


Figure 33.3: A snapshot of the DDLab screen, showing a 1d tcodes unbiased sample (TFO-mode) with 234224 tcodes (file v8k5tm.sta), sorted by min-max entropy. *Left*: space-time pattern blocks of  $n=150 \times 100$  time-steps showing increasing sample indexes starting at index 1 pausing at index 6 (section 33.7.2). *Bottom Center*: at the pause above, enter **q** then **K** to show the scatter plot, based on the min-max entropy (the maximum up-slope), where the frequency of points falling on a  $266 \times 256$  grid is indicated by colors (section 33.6). *Top Center*: at the prompt in section 33.6, enter **q** then **f** for the 2d histogram, with the frequency of points in each grid square represented on the  $z$ -axis, in this case in log form (section 33.6.3).

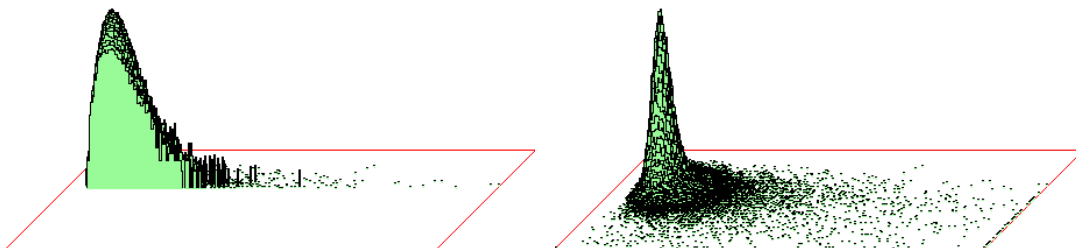


Figure 33.4: The 2d histogram in figure 33.3 can be presented according to *Left*: successive  $y$ -axis (mean entropy) slices — paused at  $y=45$ . and *Right*: a normal  $z$ -axis (frequency of rules) — the default.



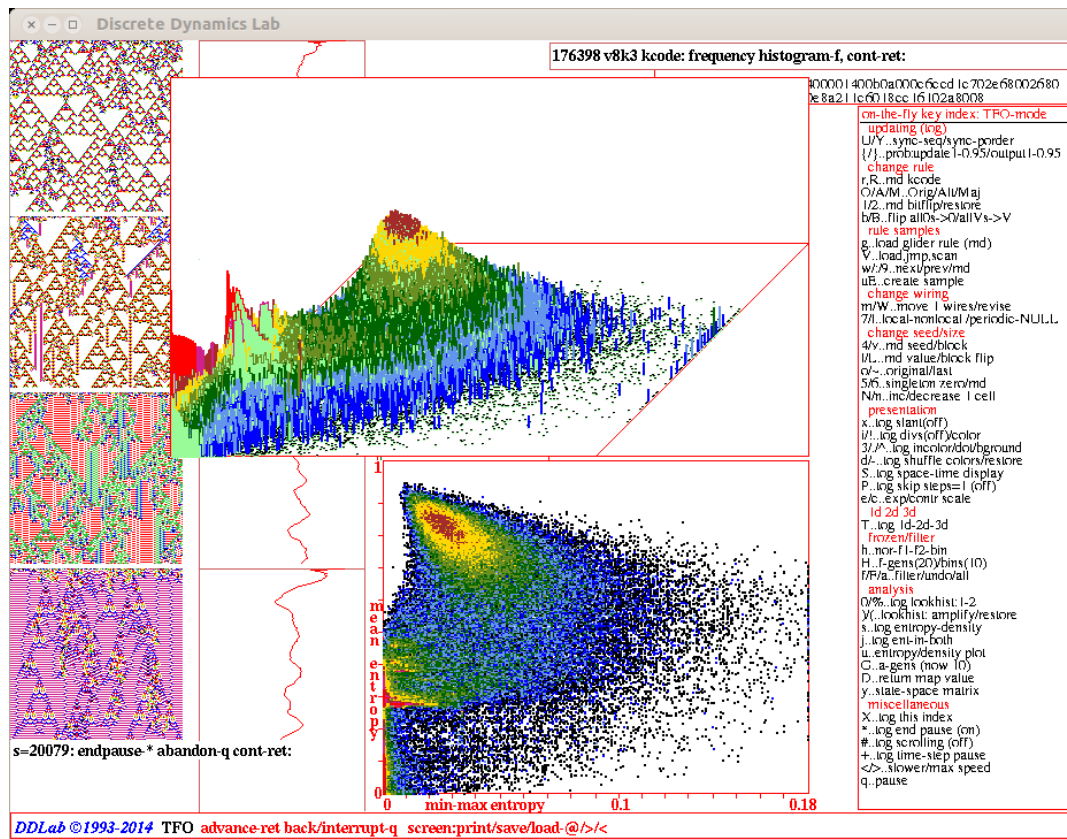


Figure 33.5: A snapshot of the DDLab screen, showing a 1d kcode sample (TFO-mode) with 176398 rules (file v8k3B50.sta), sorted by min-max entropy. The random rules were biased by setting the fraction of non-zero values in the rule-table (the  $\lambda$ -parameter) to 50% in section 16.3.1. An uneven distribution of values in the rule-table is more likely to result in complex dynamics than an unbiased rule-table (contrast with figure 33.3). This is borne out by the extensive spread of rules into the high variability area of the plot. About 14% of this rule-space, with the highest entropy variability, display particle interaction and other complex structures in space-time patterns. *Left:* space-time patterns in blocks of  $n=150 \times 150$  time-steps showing sequential examples, pausing at sample 20079. *Bottom Center:* the scatter plot with min-max entropy. *Top Center:* the 2d histogram with the  $z$ -axis in log form.

### 33.1 Input entropy and variability (min-max or sdev)

The input-frequency histogram tracks how frequently the different entries in a rule-table are actually looked up and is usually averaged over a moving window of time-steps. The input-entropy is the Shannon entropy<sup>1</sup> of this input-frequency histogram.

<sup>1</sup>The Shannon entropy of the input-frequency histogram, the input-entropy  $H$ , at time-step  $t$ , for one time-step ( $w=1$ ), is given by  $H^t = -\sum_{i=0}^{S-1} \left( \frac{Q_i^t}{n} \times \log_2 \left( \frac{Q_i^t}{n} \right) \right)$ , where  $Q_i^t$  is the lookup frequency of neighborhood  $i$  at time  $t$ .  $S$  is the rule-table size (section 13.2.1), and  $n$  is the CA size.  $H$  is usually averaged over a moving window of (say  $w=10$ ) time-steps.



One of two measures of variability of the input-entropy can be selected, either standard deviation<sup>2</sup> or a min-max measure — the maximum up-slope found so far — the rise in entropy following a minimum value (figure 33.2). When trawling for complex rules, min-max is preferable because standard deviation would give a high value to monotonic entropy decrease, characteristic of a foreground pattern gradually dying out, which would contaminate a sorted sample of complex rules. Dying out dynamics give low min-max so this problem is avoided.

## 33.2 Creating a rule sample — initial prompts

To automatically create a rule sample, or run a test, first set up any CA with the desired parameters: — SEED-mode (rcode, kcode or tcode) or TFO-mode (kcode or tcode),  $v$ ,  $k$ , and system size — 1d, 2d or 3d. The size should be relatively small (say  $n=150$  for 1d,  $40\times 40$  for 2d) to amplify variability in complex dynamics because local order and chaos tend to balance each other out in a large network. Then run the space-time patterns (chapter 32). For 1d, scrolling might be turned off (for speed) in section 32.13.3. The input-entropy plot must be active; this is the default for 1d, but for 2d or 3d it needs to be activated (sections 31.5, 32.12.3).

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 scatter plot (section 32.12.5). If this plot is current, the on-the-fly prompt is **E..create sample**, so just **E**. Alternatively, enter **q** for the space-time pattern pause interrupt (section 32.16), and enter **y** giving a small top-right prompt (which allows any on-the-fly option) enter on-the-fly option: enter **u**, then **y** again followed by **E**. The following prompts are presented in turn in a top-right window (enter **q** to backtrack), and summarized below — with more detail in the sections indicated.

**entropy sample, automatic-a test-(def): sdev-s min-max-(def):**  
**record time-steps, start (def 30): end (def 430):**  
**sample for each rule (def 5): bias rnd rules -b:**

*options ... what they mean*

**automatic-a** ... to create an automatic sample (section 33.4).

**test-(def):** ... enter **return** for a test (section 33.3), showing various measures and the entropy/density scatter plot for successive rules. At the test pause various options are also provided, including changing to automatic.

**sdev-s min-max-(def):** ... enter **return** for the min-max measure of variability, or **s** for standard deviation (section 33.1).

**start (def 30):** ... enter the time-step when measures should start. By not including the initial time-steps, the dynamics can be allowed to settle into typical behavior before measures are recorded.

**end (def 430):** ... enter time-step when space-time patterns and measures stop.

**sample for each rule (def 5):** ... enter the number of runs from different random initial seeds, from which measures are averaged.

**bias rnd rules -b:** ... to bias the random rules (section 33.2.1).

<sup>2</sup>The standard deviation is given by  $\sigma = \sqrt{\sum \frac{x_i^2}{n}}$  where  $x_i$  is the deviation from mean input-entropy at time-step  $i$ , and  $n$  is the number of time-steps measured.

### 33.2.1 Biasing random rules

A rule bias that was applied when setting a single rule in section 16.3, either a density-bias (section 16.3.1), or a value-bias (section 16.3.2), will carry over to the random rule sample. A density-bias (the proportion of non-zero values in the rule-table —  $\lambda$ -parameter) is always present in the background, but if not deliberately altered gives an equal probability of each value — effectively no bias. An active value-bias (the proportions of each value in the rule-table) will override the density-bias.

Enter **b** in section 33.2.1 to reset the value-bias, and other biases — isotropic rules, and adapting the rule-table so that the all-zero neighborhood outputs zero, or more generally all- $V$  neighborhoods output  $V$ . A top-right prompt is presented, which varies according to context. The first bias option (not in TFO-mode) is to set an isotropic rule where rotated and reflected neighborhoods (in 1d, 2d and 3d) have the same output. Rules in TFO-mode are totalistic, so isotropic by definition.

**iso-i:** *(not for TFO-mode because totalistic rules are isotropic)*

If the isotropic option was not selected, options for the value-bias follow (if in TFO-mode these are the first options). The options depend on whether a value-bias is active or not,

*if a value-bias is active — the values are shown, in this example for v8k3*

**value-bias, (p)7-0: 44 8 16 4 8 4 8 8 cancel-c set-v keep-(def):**

*if a value-bias is not active*

**value-bias, set-v:**

For an active value-bias, enter **c** to cancel (deactivate), or **return** to keep.

Enter **v** to set (or reset) the value-bias — this results in a series of prompts, firstly to set the value bias either as percentages, or as actual numbers of each value (if the rule-table size  $S \leq 255$ ) with the prompt **%/value-p/v:** or **%/value-p:**. If **p** (or **v** if applicable) is entered, prompts will follow to set the percentage (or the number) of each value from 0 to  $v-2$ ; the value for  $v-1$  is set automatically. The method is described in greater detail in section 16.3.2.

Following the value-bias, or if value-bias is cancelled, kept, or not set/reset, two final successive prompts are presented as follows,

**all0s-0: allVs-v:**

Enter **0** to set the output of the all-0s neighborhood to 0. Enter **v** to set the outputs of all uniform neighborhoods to the neighborhood value (the same can be done on-the-fly, section 32.5.6). These options, which provide a stable background domain, apply on top of previous biases. If **q** is entered, the prompts revert to the first bias option — this can be useful to reconfirm the value-bias.

## 33.3 Running a test

Enter **return** at the first prompt in section 33.2 to run a test, then enter **return** to accept other parameter defaults, or change any of these if required.

Space-time patterns will run from different random initial states for the set sample of each same active rule (without pause until the sample is complete), and for the set number of time-steps. Initially, the active rule will be the current rule, thereafter a new random rule can be applied

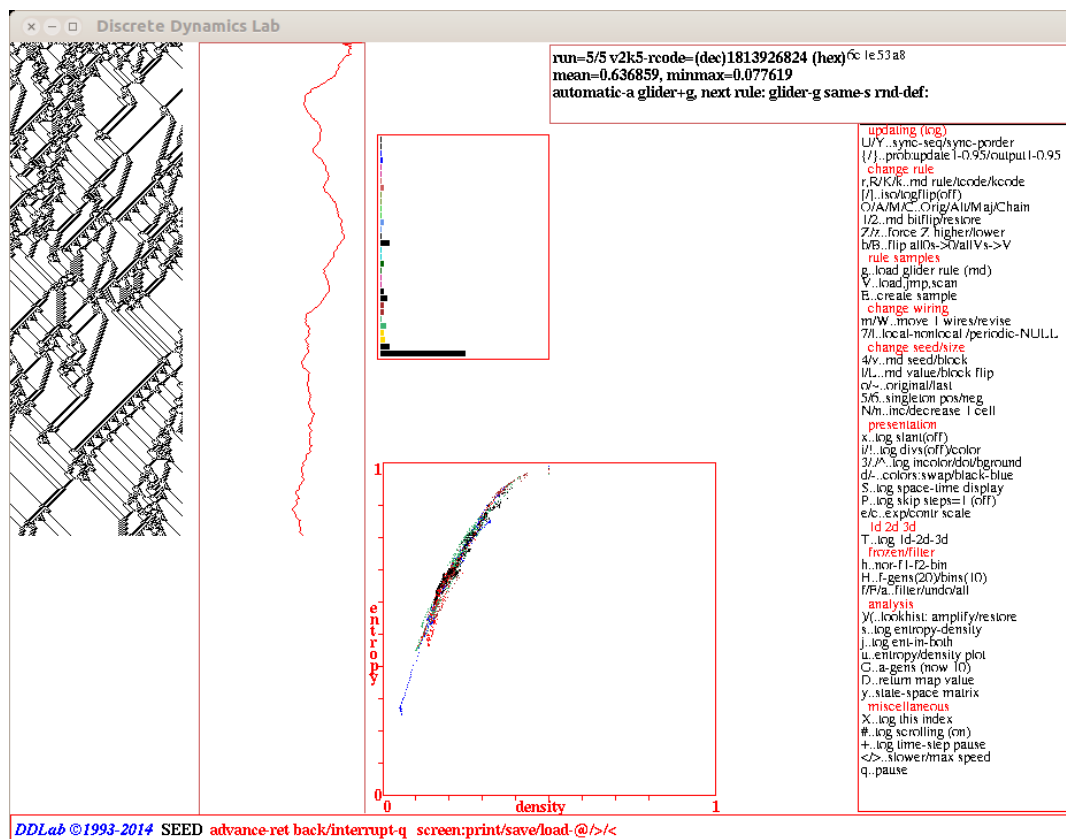


Figure 33.6: A snapshot of the DDLab screen while running a test, showing mean entropy and min-max entropy in a top-right window for a sample of random initial states. This example for a 1d  $n=150$   $v2k5$  CA, rcode  $6c1e53a8$  from the glider rule collection (section 3.6.1). As space-time patterns (including the input-frequency histogram and input-entropy plot, section 31.5) are being drawn, the entropy/density scatter plot (section 32.12.5) 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 in section 33.2 (start 30, end 430, size 5) were used, meaning the measures were started at time-step 30 and ended at 430 (401 time-steps), and the number of runs from different random initial states was 5.

according to any biases described in section 33.2.1, or the same rule can be repeated, or a rule from a glider rule collection (section 3.6.1) can be set. The run can also be converted to automatic.

### 33.3.1 Test data

After each run from an initial state in the sample, the following top-right window with data on the rule and measures is presented (for example),

```
run=5/5 v2k5-rcode=(dec)1813926824 (hex)6c1e53a8
mean=0.673505, minmax=0.163594 (or sdev)
```

This remains unchanged until the next sample is complete.

*data ... what they mean*

- run=3/5** ... the current sample (e.g. sample 3) out of the set sample of 5.  
At **run=5/5** there will be a pause for further prompts, see below.
- single=33** ... if single cell input-entropy was set in section 31.5.1 the cell index will be indicated.
- v2k5-rcode=** ... the rule in decimal (if applicable), and in hex.
- mean=** ... the mean entropy.
- minmax=** ... the min-max entropy, alternatively (**sdev=**) the standard deviation if this was selected in section 33.2.

While a run is in progress, the entropy/density scatter plot (section 32.12.5) is drawn in a lower center window. The plot shows distinctive signatures for complex rules, and is drawn in alternating colors for successive sample runs.

### 33.3.2 Test options

When the sample runs are complete (i.e. **run=5/5**, figure 33.6), the top-right window will pause to include the following options,

**automatic-a glider+g, next rule: glider-g same-s rnd-def:**

*options ... what they mean*

- automatic-a** ... abandon the test and initiate an automatic rule sample (section 33.4).
- glider+g** ... enter **a** followed by **g** for an automatic sample as above, but limited to a glider rule collection (section 3.6.1).
- next rule:** ... continue the test for the next rule which can be selected as follows:
  - glider-g** ... to test a rule from a glider rule collection (section 3.6.1).
  - same-s** ... to re-test the same rule (with different random initial states).
  - rnd-def** ... enter **return** to test a new rule set at random, or with a bias (section 33.2.1).

## 33.4 Creating an automatic rule sample

Enter **a** in sections 33.2 or 33.3 to create an automatic sample. A filename for the sample (**.sta**) will first be selected in a top-right window (default filename **my\_stat.sta**, section 35.3), 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:**  
(or the selected filename)

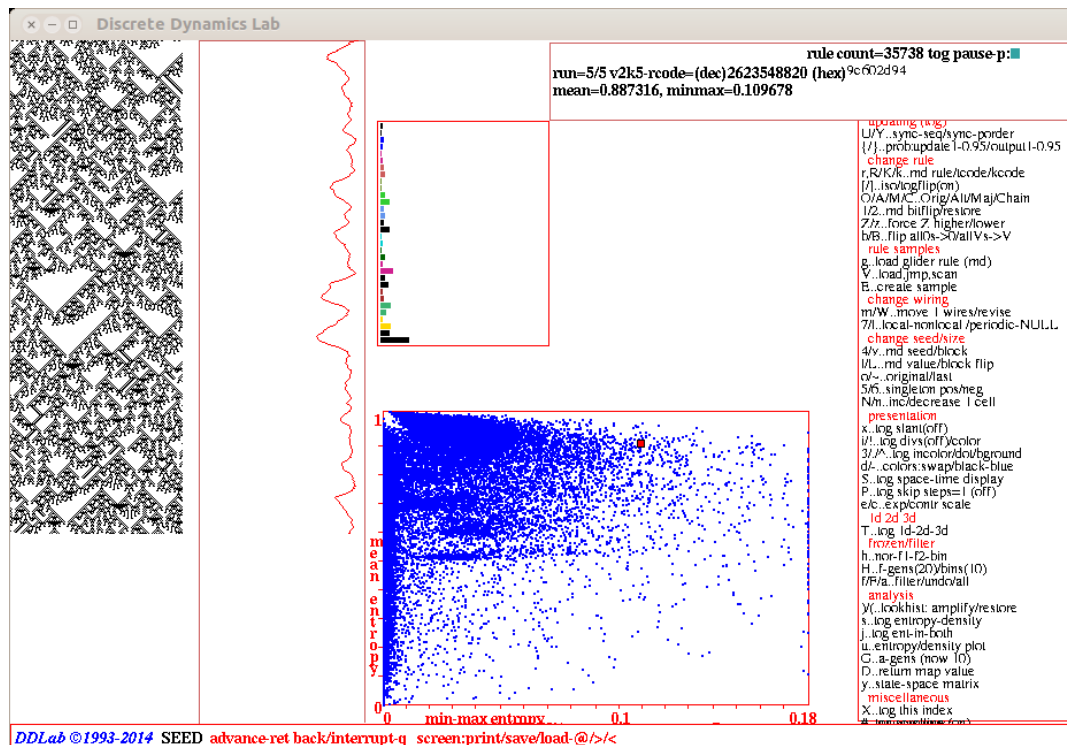


Figure 33.7: A snapshot of the DDLab screen while creating an automatic sample — this example for 1d  $n=150$   $v2k5$  isotropic CA. The sample results are shown in figure 33.8. The default parameters in section 33.2 were applied, except that the sample is biased for random isotropic rules. The sample was paused at rule-count 35738, at rcode(hex)9c602d94. The scatter-plot, mean entropy against entropy variability, is continuously updated in a lower-center window. After the runs for each rule are complete, a small red square is plotted. This is replaced by a blue dot when the next rule is complete, and so on. Information about the last rule is shown in a top-right window. Note that any variability above 0.18 is rounded down to 0.18

To pause after each new rule, enter or add  $+$ , i.e.  $a+$  to append data and pause. The automatic sample will then start, plotting the mean entropy against the entropy variability, one point for each new random rule in the scatter plot in a lower-center window (figure 33.7). After 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 following top-right window keeps track of the count of rules so far, with a reminder that a pause can be set on-the-fly with key  $p$ .

**rule count=1457 tog pause-p**

Initially there is no pause between each new rule, however if a pause is activated with on-the-fly key  $p$ , at the pause the top-right window will include the data as in section 33.3, for example,

**rule count=24569 tog pause-p:**  
**run=5/5 v2k5-rcode=(dec)3935933814 (hex)ea 99 95 76**  
**mean=0.673505, minmax=0.163594 (or sdev — as in figure 33.7)**

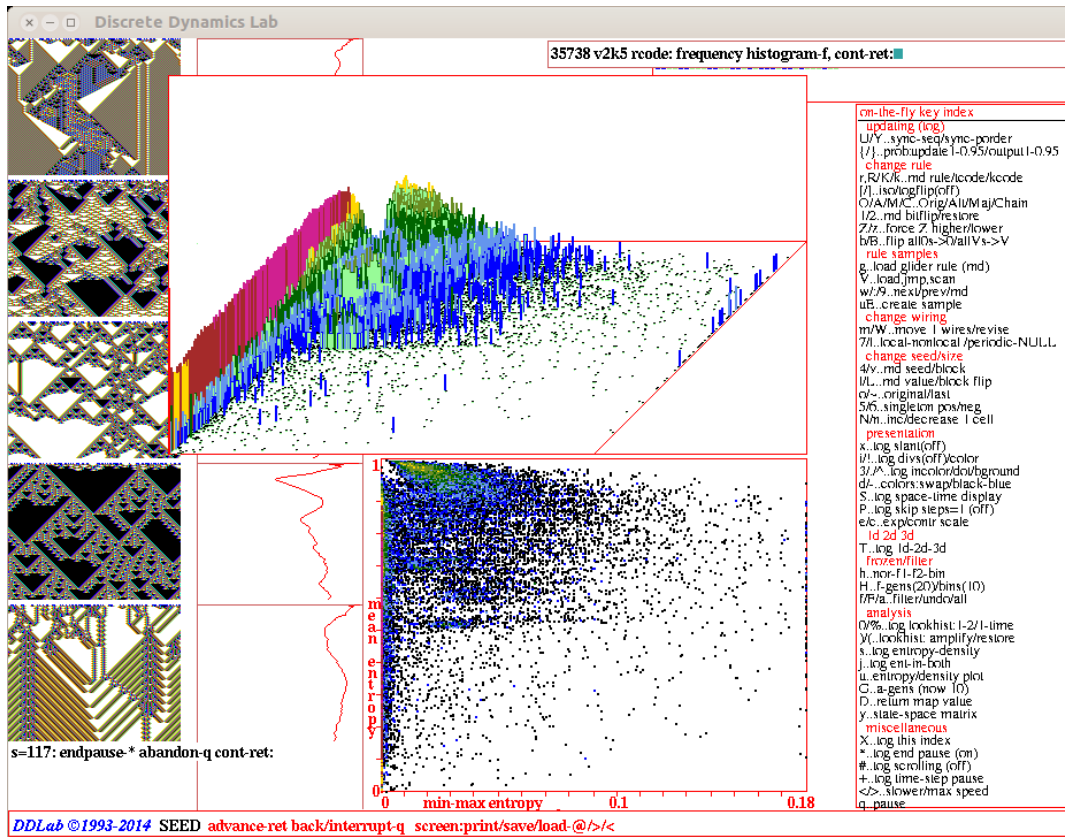


Figure 33.8: A snapshot of the DDLab screen after loading an automatic sample of 35738 rules, for 1d  $n=150$   $v2k5$  isotropic CA, created in figure 33.7 (file `v2k5iso1.sta`). The sample was sorted by min-max entropy. *Left*: space-time pattern in blocks of 120 time-steps showing sequential examples, pausing at sample index 117 (section 33.7.2). *Bottom Center*: the scatter plot with min-max entropy. *Top Center*: the 2d histogram with the  $z$ -axis in log form.

If the pause is set, enter **return** (or **p** to toggle the pause off) to continue with the next rule. The sample will continue indefinitely until paused with **p**, or interrupted with **q** (section 32.14 — enter **q** again to backtrack). If interrupted, the sample can always be continued later by appending a new sample run to an existing filename. To considerably speed up computation, turn off the space-time graphics with on-the-fly toggle **S** (section 32.9.8). Note that most on-the-fly options such as “presentation” (section 32.9) and “frozen/filter” (section 32.11) will work while the sample is in progress, but other on-the-fly options should be used with caution as the consequences are unpredictable.

### 33.5 Loading, sorting and displaying a sample

To load an automatic sample, enter on-the-fly key **V..load,jmp,scan** while space-time patterns are running, or when interrupting space-time patterns (section 32.14) enter **E** to load, or **K** to “keep” a sample that is currently active (section 32.16.2).

One of the following top-right prompts are presented, depending on whether a sample is currently active,

**entropy variability, sdev-s, min-max -(def):** *(if a sample is not active)*

**rule sample: load new-n, rule index/scan 1-24589 (1-def):** *(an active sample, as figure 33.7)*

If a sample is active, enter **n** for a new sample, or enter **return** or a number **x** to scan space-time patterns of successive rules from rule index 1 (the default), or from rule index **x**, in the current he sample (section 33.7.2).

If loading a new sample (section 35.3), note that any sample (**.sta**) file will load, so a file where  $[v, k]$  conflicts with the base network will result in a meaningless scatter plot (section 33.10). Once the file is loaded, the following top-right list/sort/display prompt is presented,

**list: all-l patch-p coords-c, sort-s backtrack-q (sublist-S save-SS)** *(patch-p if a patch is active)*

**min-max plot: Z-Z lda-L ent-(def) smalldots+d:** *(Z and L for rcode only)*

*options ... what they mean*

**list:** ... to list the sample (sections 33.8), which allows selecting a rule from the list (section 33.8.1).

**all-l** ... to list the sample starting from index 1.

**patch-p** ... if a patch was predefined in section 33.6.2, to list the sample confined to rules within the patch starting from lowest index.

**coords-c** ... to list the rules at, or near to, given scatter plot coordinates (section 33.9), also done by probing with the mouse (section 33.6.1).

**sort-s** ... to sort the sample — necessary for an unsorted sample for effective listing or scanning space-time patterns. A prompt for a new filename (**.sta**) for the sorted file will be presented (section 35.3), which can be the same as before. The rules are sorted firstly by decreasing variability (min-max or sdev), then by decreasing mean entropy for each measure of variability — for a large sample this might take a little time.

**backtrack-q** ... to backtrack to showing space-time pattern, keeping the active sample.

**(sublist-S save+S)** ... to see or save a sublist of rules<sup>3</sup> created from an existing sample.

**min-max plot:** ... (or **sdev plot:**) show the scatter plot (variability, min-max or sdev on the  $x$  axis) in a lower center window (section 33.6), which can then be “probed” with the mouse (sections 33.6.1) to list and select rules.

**Z-Z** ... *(rcode only)* to plot variability against the  $Z$  parameter (figure 33.13 *Top*).

**lda-L** ... *(rcode only)* to plot variability against  $\lambda$ -ratio (figure 33.13 *Bottom*).

**ent-(def)** ... *(the default)* enter **return** to plot variability 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) “small dots” is the default, so the prompt reads **bigdots+d**.

---

<sup>3</sup>This function involves modifying and recompiling DDLab’s code. A sublist of selected items from a pre-existing **.sta** file can be included in code to create a new sample. To see the sublist as in sections 33.8 or 33.6 enter **S**. To also save the sublist as a new **.sta** file (enter **SS**). The sublist is inserted in function `show_list_rules()` in file `forvarx.c`.



Once the scatter plot is displayed, further options include probing the plot with the mouse and listing/selecting rules (section 33.6.1), and displaying a 2d frequency histogram, where the vertical axis represents rule frequency.

## 33.6 The rule sample scatter plot

Enter **return** in section 33.5 to display the scatter plot of mean entropy against entropy variability (min-max or sdev) in a lower center window. For rcode only, **Z** or **L** can be entered for alternative plots, where mean entropy is replaced by either the *Z*-parameter (figure 33.13 *top*), or by  $\lambda$ -ratio (figure 33.13 *below*).

Rules are represented by colored square points on a 256×256 grid, and the scatter plot color scheme indicates the frequency of rules falling on each square (figure 33.9).

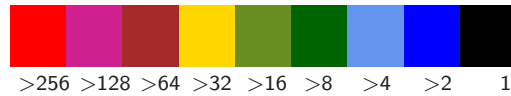


Figure 33.9: The scatter plot color scheme indicates the frequency of rules falling on each square of the 256×256 grid. Unoccupied squares remain white.

Scatter plot reminders/options appear in the following top-right prompt,

**select rules - point-click mouse button: probe-left, patch-right  
frequency histogram or backtrack -q:**

Click on or near a point with the left mouse button to “probe” the scatter plot (section 33.6.1), or click twice with the right mouse button to define a rectangle containing rules to be examined (section 33.6.2). Enter **q** to show the scatter plot as a frequency histogram (section 33.6.3), or to backtrack to the list/sort/display prompt (section 33.5) where further backtracking restores space-time patterns.

### 33.6.1 Probing the scatter plot with the mouse, and selecting rules

The mouse pointer changes direction (to North-West) within the scatter plot, and a small top-center window shows the pointer coordinates, for example `x=84 y=213`. Clicking the left mouse button anywhere in the plot produces a top-right window (section 33.9) 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 — also useful to capture an isolated rule from a shaky mouse click. A rule can be selected from the list, and its space-time patterns run, as described in section 33.8.1. Enter **q** (or click again) to delete the list, and click the plot again at a new position to show a new list. The same list can be produced by setting the coordinates (section 33.9).

### 33.6.2 Defining a scatter plot patch with the mouse or keyboard

The last two right mouse button clicks anywhere in the plot will define a rectangular patch of rules. Each right click produces a top-right inset box, for example,

First: `patch: x1=130 y1=84, activate-p`

Second: `patch: x1=157 y1=69, activate-p`



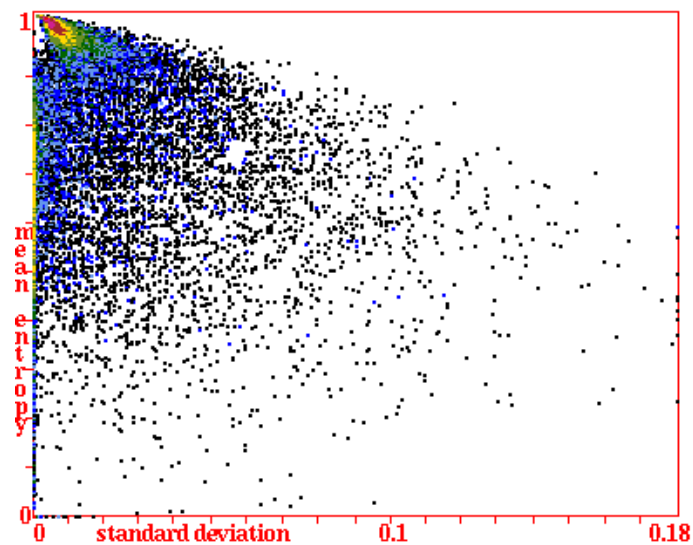


Figure 33.10: The rule sample scatter plot, showing a 1d rcode sample with 17680 unbiased random rules (file `v2k5ss.sta`), sorted by standard deviation entropy. A color scheme indicates frequency. To select a rule anywhere on the plot, click on a point with the left mouse button for a list, then select the rule (sections 33.9, 33.5).

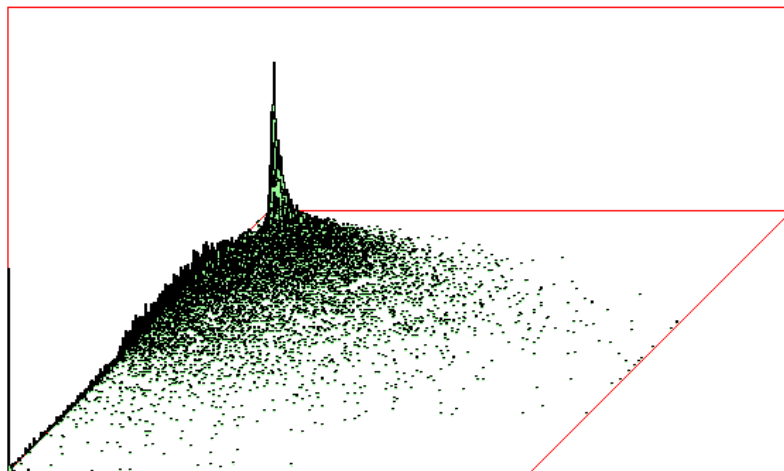


Figure 33.11: The 2d frequency histogram of the rule sample scatter plot in figure 33.10. The vertical axis is the frequency of rules falling on each square of the grid — here 256x256.

Whatever order or position of the last two right clicks, the patch will be defined by its top-right (high values) and bottom-left (low values) corners. Once satisfied with the two positions, enter **p** to activate the patch – the following top-right prompt is presented,

**patch coords (def 130 83 157 69) keyboard-k accept-def:** *(for example)*

Enter **return** to accept and activate the default patch as set previously by right mouse clicks and shown in the prompt. Alternatively enter **k** to enter coordinates (**x1**, **y1**, **x2**, **y2**;) from the keyboard — then the following additional prompt is presented,

**keyboard: (max 255x255) x1: y1: x2: y2: (presented in order)**

Enter the coordinates in any order, or enter **return** to accept the listed defaults. The patch is activated when entries are complete.

Once activated, rules in the patch can be listed (*patch-p* in section 33.5), and space-time patterns can be perused on-the-fly — key **w** for the next sample index, and key **:** (colon) for the previous index (section 33.7.1) will select rules only from within the scatter-plot rectangle. The index order is the same as specified in section 33.8. Information on the new rule, including the sample index, appears in a top-right window when the key is hit as in figure 33.14.

### 33.6.3 The scatter plot as a 2d frequency histogram

The 2d frequency histogram of the scatter plot includes a vertical axis showing the frequency of rules falling on each square of a notional grid laid over the scatter plot — the maximum resolution of this grid,  $256 \times 256$ , can be reduced.

If **q** is entered in section 33.6 the following top-right prompt is presented,

**17680 v2k5 rcode: frequency histogram-f, cont-ret: (for the sample in figure 33.10)**

Enter **return** to skip the histogram and backtrack to the prompt in section 33.5, or enter **f** to select the histogram (e.g. figure 33.11), which will appear in a top-center window — the following top-right prompts are presented in sequence to specify the presentation and other options,

**freq hist, subdiv (16,32,64,128,256-(def):**  
**save data-d, step-s, and show%-S: freq log-l: grid-g togcolor (now on)-c:**

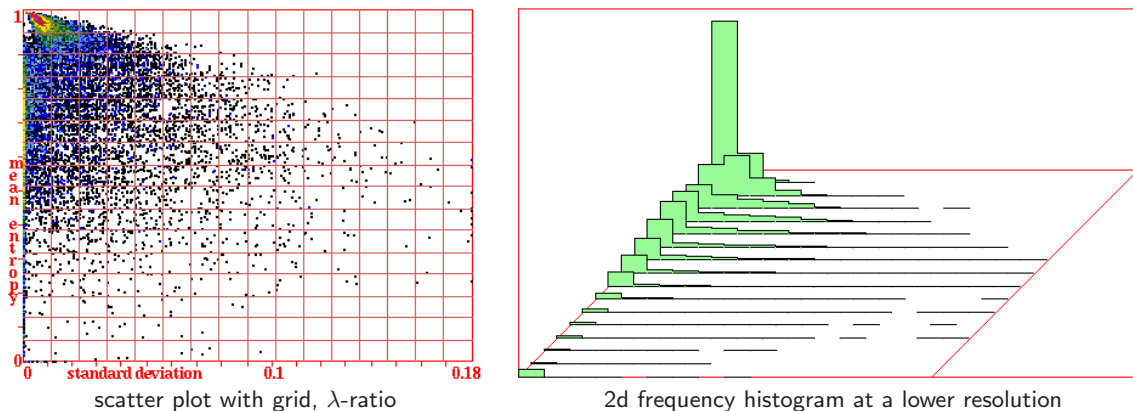


Figure 33.12: *Left*: The rule sample scatter plot — the same as in figure 33.10, but with a  $16 \times 16$  superimposed grid — by entering *grid-g* in section 33.6.3. *Right*: The 2d frequency histogram of the rule sample scatter plot in figure 33.10 redrawn at a lower resolution,  $16 \times 16$ , selected in *subdiv* in section 33.6.3. The vertical axis is the frequency of rules falling on each square of the grid. This figure also illustrates pausing after each row with *step-s* or *show%-S* in section 33.6.3.

*options ... what they mean*

**subdiv (16,32,64,128,256-(def): ...**

... The vertical axis of the 2d histogram indicates the frequency of rules falling on each square of a notional grid laid over the scatter plot. The resolution of this grid can be  $16 \times 16$ ,  $32 \times 32$ , etc. Enter **return** for the default  $256 \times 256$  grid, or enter 16, 32, 64, or 128, for a different resolution.

**save data-d ...** to save the frequency data to an ASCII file (**.dat**), laid out in rows and columns according to the grid resolution above, giving the number of rules falling on each square of the grid. For example, for a  $16 \times 16$  grid according to figure 33.12, the data is as follows,

```

4240 774 75 4 0 0 0 0 0 0 0 0 0 0 0 0
826 1062 496 173 52 12 2 1 0 0 0 0 0 0 0
644 358 260 201 120 53 15 9 3 0 1 0 0 0 0
822 242 186 152 131 60 39 24 12 3 0 0 0 0 0
869 202 156 115 91 46 44 33 15 12 5 3 0 0 0
751 139 98 81 59 55 39 22 20 10 9 2 4 1 0
578 121 103 61 43 43 35 24 26 14 6 7 3 2 1 2
474 100 88 53 35 34 23 22 18 10 5 4 5 2 2 1
384 77 53 36 28 18 15 9 6 6 3 1 3 2 1 2
156 49 39 21 29 17 15 7 7 6 3 2 0 0 0 5
103 24 22 20 15 8 8 5 3 8 4 4 0 1 1 2
68 16 10 10 7 2 8 5 5 3 0 1 0 0 1 0
59 6 10 4 6 3 3 2 1 4 0 3 0 0 1 1
40 1 4 5 3 4 0 2 1 0 0 0 0 0 0 0
36 2 4 3 2 1 1 0 0 0 0 0 0 0 0 0
214 4 1 2 0 1 3 0 1 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, then enter **return** for each next row. If **S** is entered, the height data for successive rows is displayed as a fraction of the maximum height, in a top-right window.

**freq log-l: ...** for a log scale of the vertical axis.

**grid-g: ...** to overlay a grid, selected in **subdiv** above, on the scatter plot, as in figure 33.12.

**togcolor (now on)-c ...** enter **return** to color histogram columns according to colours in the scatter plot, or **c** for monochrome.

Once these options are complete, the 2d histogram of the scatter plot is drawn in a top-center window, and the histogram prompt in section 33.6.3 reappears. Enter **f** to reselect the histogram, or **return** (or **q**) to backtrack to the prompt in section 33.5 — further backtracking will restore space-time patterns.

## 33.7 Scanning sample space-time patterns

If a sample is currently active, space-time patterns, in any presentation (1d, 2d or 3d), can be scanned from the sample. For more meaningful results the sample should first have been sorted in section 33.5.

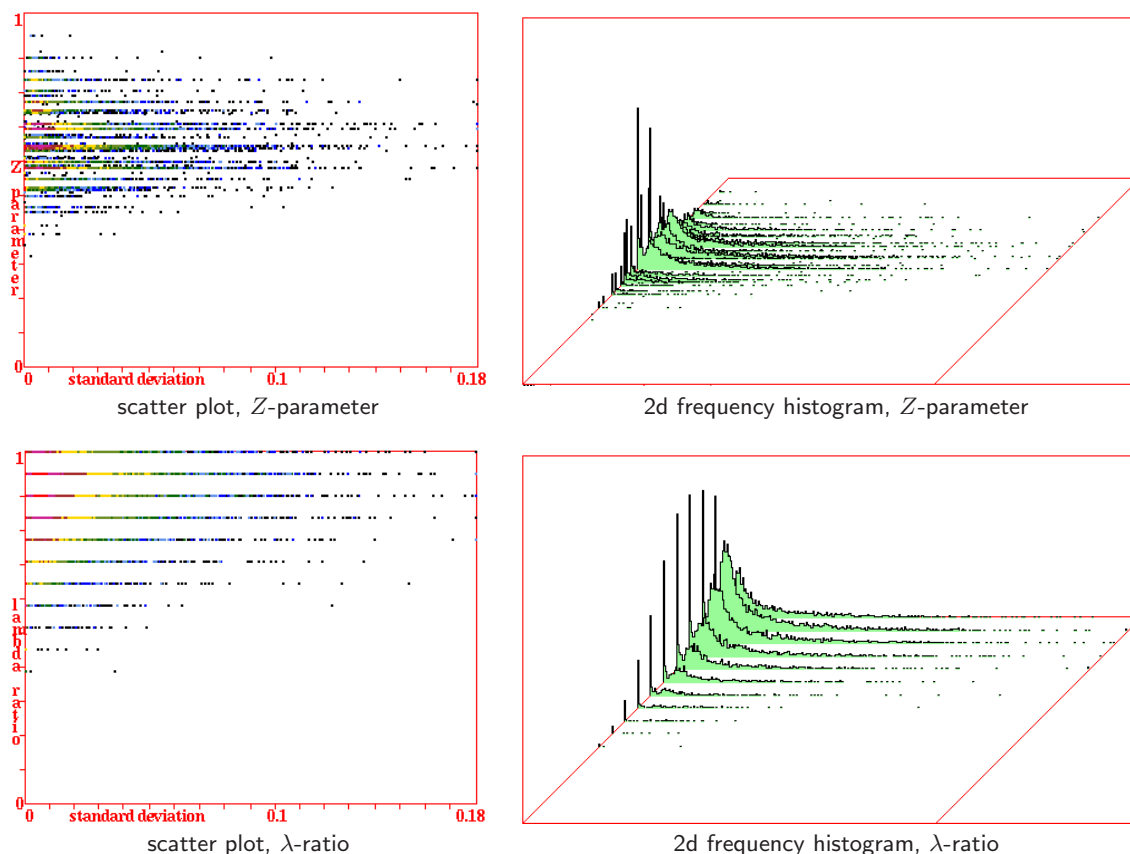


Figure 33.13: Alternative scatter plots and 2d frequency histograms (rcode only) where mean entropy is replaced by: *Top*: the  $Z$ -parameter, and *Bottom*:  $\lambda$ -ratio, for the same rule sample as in figure 33.12. Listing by plot coordinates or probing with the mouse apply as usual (section 33.9).

### 33.7.1 Scanning on-the-fly

The simplest method of scanning space-time patterns on-the-fly is with the following keys,

**w**/:9..next/prev/rnd

Enter **w** for the next sample index, enter **:** (colon) for the previous index — the sample index order is as specified in section 33.8. If a patch is active (section 33.6.2) each next or previous sample index is restricted to rules within the patch, which recycle on reaching the ends. Enter **9** for a random index anywhere in the sample. Each rule change also sets a random initial state, and space-time patterns continue until a new on-the-fly key hit, or if interrupted. Information on the new rule, including the sample index, appears in a top-right window when the key is hit as in figure 33.14.

To start scanning from a selected rule index, enter on-the-fly key **V..load,jmp,scan**, then enter the rule index at a top-right prompt, for example,

**rule sample: load new-n, rule index/scan 1-24589 (222-def):** (sample in figure 33.8)

```

kcode v4k7: s=53 me=52 mm=159 patch-OFF
.....
hex=c03034 1c9b4403016100628330144c97
1dcf83008080232b98023183621c

```

Figure 33.14: Information that appears in a top-right window when scanning sample space-time patterns with on-the-fly keys `w/:/9`, showing  $[v, k]$ , the sample index ( $s=10$ ), the mean entropy (me), the rule's mean-entropy (me) and variability, standard-deviation (sdev) or min-max (mm), and the rule as a bit/value-pattern and in hex (from the sample in figure 33.16).

### 33.7.2 Scanning automatically in blocks of time-steps

To scan automatically in equal blocks of time-steps (time-blocks), or to just start from a new rule index, enter on-the-fly key `V..load,jmp,scan`, which gives the following top-right prompt, for example,

```
rule sample: load new-n, rule index/scan 1-176398 (17-def): (sample in figure 33.5)
```

Enter a number for a new rule index, or `return` for the current index which is shown (or `n` for a new sample). For example, if 33 is entered for the rule index — the following prompts allow scanning automatically in blocks of time-steps,

```
scan time-blocks index from 17: incr-i, keep-k no-blocks-def:
```

Enter `return` to reject time-blocks and continue from the selected index, Enter `i` to scan time-blocks automatically by increasing indexes. If a patch is active (section 33.6.2) successive indexes are restricted to those within the patch and recycle on reaching the end. Enter `k` to scan in time-blocks but keeping the selected index, changing only the initial state.

If time-blocks were selected with `i` or `k` above, a top-right prompts are presented as follows,

```
set time-block (def 150): seedtype: block-v single-5/6 (def-all):
```

Firstly set the number of block time-steps, then select the type of random initial state (the seed) for each new time-block. The default is a random seed over `all` the space; `block-v` sets a central random seed-block with dimensions as set in section 21.3. These random seeds retain the density-bias set in section 21.3.2. The `single-5/6` option sets a singleton seed — for  $v=2$  `5/6` gives a positive/negative singleton seed surrounded by the opposite value – for  $v \geq 3$  `5` gives a non-zero random value against a uniform background of zeros, and `6` gives a random value against a uniform background of a different random value.

Following these options space-time patterns in time-blocks will run. Note that the type of seed can be reset on-the-fly with key-hits `all-4` and `block-v` section 32.8.1, and `single-5/6` with key-hits `5` or `6` section 32.8.4 and 32.8.5.

### 33.7.3 Scanning 1d samples

For 1d space-time patterns, including 2d and 3d samples shown in 1d, if scrolling is set to `off`<sup>4</sup> there will be a gap between each block, but no gap otherwise. To pause after each screen-full of

<sup>4</sup>Scrolling can be toggled *on/off* in section 31.2.6, or on-the-fly (`#..tog scrolling`, section 32.13.3).

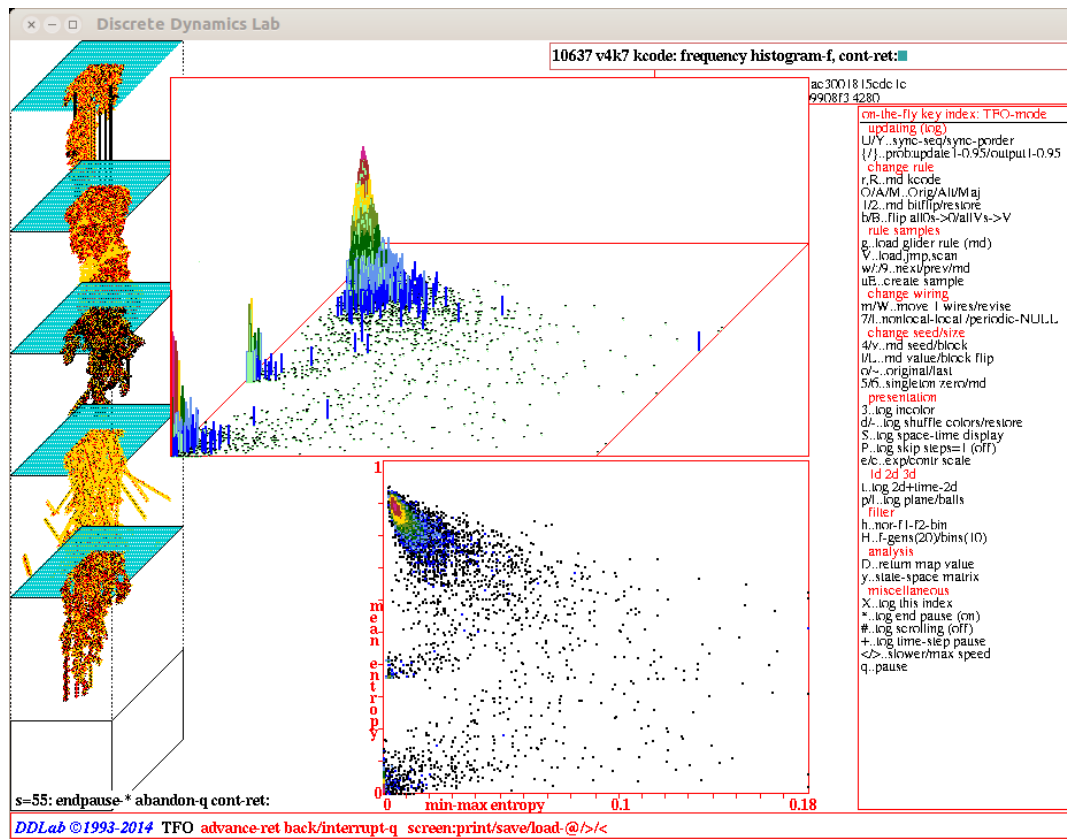


Figure 33.15: A snapshot of the DDLab screen, showing a 2d kcode sample (TFO-mode) with 10637 rules (file v4k72d52.sta), sorted by min-max entropy. The random rules have a percentage value-bias 3-0: 52 16 16 16, and outputs of all uniform neighborhoods to the neighborhood value (section 33.2.1), resulting in a greater frequency of complex rules than in an unbiased sample. *Left*: 2d space-time patterns  $88 \times 88$  in blocks of 100 time-steps, paused at rule index 24. *Bottom Center*: the scatter plot with min-max entropy. *Top Center*: the 2d histogram with the  $z$ -axis in log form.

blocks, set the end pause *on*<sup>5</sup>. The pause occurs when no more blocks will fit, with the following bottom-left prompt, which is presented below the last block, as in figure 33.15 and other DDLab screen snapshots in this chapter,

**s=24: end pause-\* abandon-q cont-ret:** (as in figure 33.15)

The rule index of the last block is given by **s=24**. Enter **return** twice for the next set of blocks — the interrupt prompt (section 32.13.4) is presented first (showing the next rule) — here other options can be activated including on-the-fly options (*on-the-fly-y*, section 32.16.9) where the scatter plot and its 2d histogram can be added to the presentation (as in figures 33.3, 33.5, 33.8, 33.15). Enter **\*** to undo the block end pause, **q** to abandon blocks and backtrack.

<sup>5</sup>Pausing 1d space-time patterns when scrolling is *off* can be set on-the-fly (**\*.tog end pause (on)**, section 32.13.2), and also in section 31.2.7, or from the interrupt prompt (section 32.13.4).

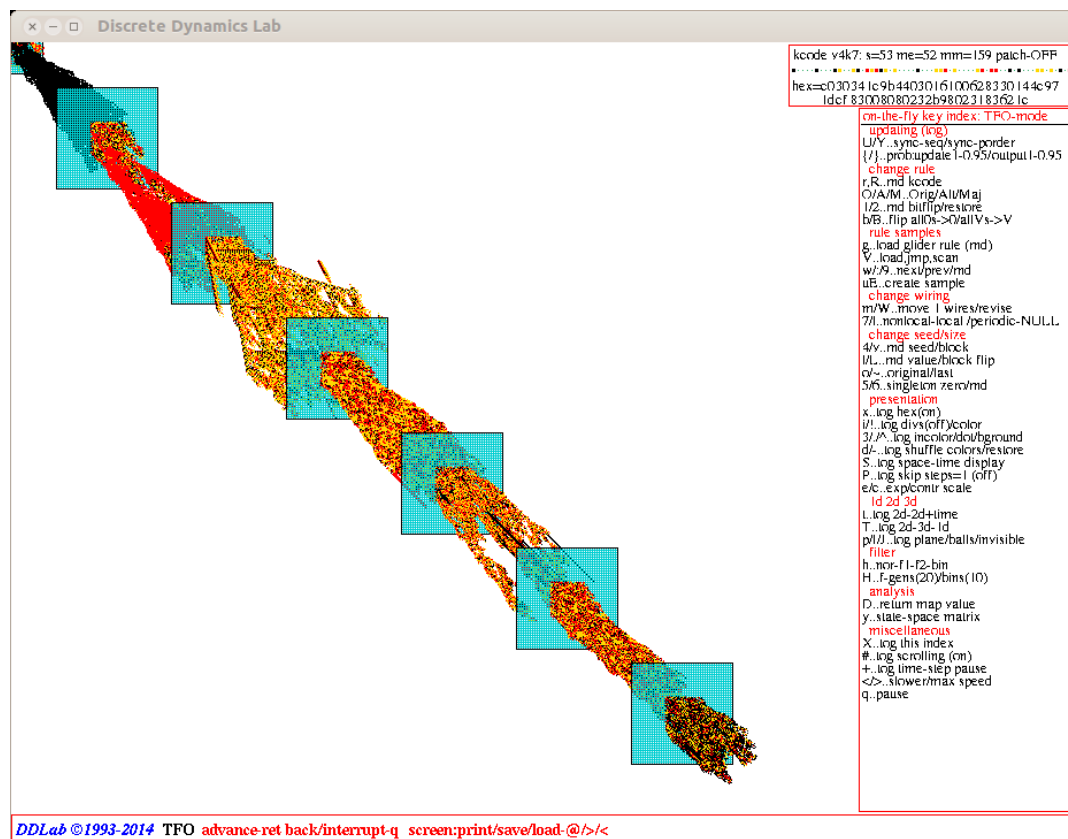


Figure 33.16: A snapshot of the DDLab screen while scanning a 2d CA  $88 \times 88$  in blocks of 100 time-steps scrolling diagonally, for biased *v4k7* kcode in figure 33.15.

### 33.7.4 Scanning 2d or 3d samples

A sample created for a 2d or 3d CA can be scanned as described in sections 33.7, with the difference that a 2d or 3d space-time pattern is presented initially as a “movie” of successive time-steps. However, a 2d presentation can be shown as a space-time pattern with a historical time dimension in either vertical or diagonal time-steps, where a rule sample can be scanned automatically in time-blocks (section 33.7.2).

For vertical 2d time-steps (on-the-fly key **t..tog 2d-2d+time**, section 32.10.2) scrolling can be either *on* or *off* (on-the-fly keys **#..tog scrolling**) including setting a pause (figure 33.15).

For diagonal 2d time-steps (on-th-fly key **#..tog scrolling** from a 2d “movie”) as in figure 33.16 scrolling is always active. Note that whatever the underlying CA dimension, 1d, 2d or 3d, the presentation dimension is interchangeable on-the-fly with key **T**.

### 33.8 Listing a sample

10637 v4k7 rules mean minmax			
1	154	255	c9180cc01f809140412dd08842ccc
2	150	255	c8202120e04140b004ec18770d0028
3	127	255	e8010223bd01834003a04c2c410560
4	127	255	c02ec9fd41c003c040e9f08009a553
5	76	255	daf4002c098ec40fe940c300034810
6	15	255	d120c323032020320e b460604bf44d
7	5	255	ecfa b2f060408404e00204c41d830e02
8	109	237	dc700c602012f0e703508c478000c2
9	140	235	c0860406e701937010c251040ec740
10	77	234	ce33000c0e011235001c020831bd8d
11	98	232	c09d32c011f016038ce0c83d04006b
12	160	218	c1b07003d40f302c65d004f9f840
13	144	217	df83004e4c4100120c7070a46805
14	84	216	f3d703000031500c72c4d3fa02d34
15	120	214	f439f009481c0e05d10cb0e83303
16	27	210	d030d03062e70f450b4220341f0776
17	23	202	c8103353dc3c0b8d231000d43c13649
18	127	202	ec06cbea7084f40810285c8078f4
19	37	197	f70d095838c00c0920251000634710
20	75	196	f811cd80031c0cb80312a005e4b48
21	107	196	c07af0c2214a0103c0261cd43c0a0
22	114	192	d3505944fe0ea0c2202135b02640
23	10	191	f4e2184900800922d014bd0e003020
24	165	190	c300778e2641c1428c1284344200f

more-ret jump-j quit/select-q;  
enter rule index 1-10637:99■

default listing from index 1, and  
requesting a jumping to 99

10637 v4k7 rules mean minmax			
99	114	128	c004a30c043e223c00483eac1820
100	128	127	eee3410880b014f200c3086430
101	178	127	f0c98e041d8c303000049e1840
102	147	127	e30b25201ec819c8243d46d034
103	140	127	e40127d83008f003185001c6e8
104	28	126	e01b44c80058c18c02179c200c0c
105	10	126	efb1c34c6ef621314c0023090020
106	91	126	c0fc1ad20843035c400a08602e33
107	85	125	c6c004a107ec360d61c0025b02fb
108	156	125	c2571690f0e0874038c8bcd402
109	156	124	f108b8812830ab4e0b143ef051
110	133	124	ff5e20228ecc820d243038420
111	138	124	cdc4e11c004cb80c926040800c
112	166	124	cc39803c04845103299061e000
113	139	124	c41c81000b3018b2a3b8100300
114	165	123	fd0240ece24b342c94440e4f80
115	35	121	e37008a2001c42380811160b82817
116	173	121	c8478f028462d990300c3dec00
117	117	121	ee8eb10880c308c4c07ec450c1
118	27	120	e0d024508c2d8045748080d20370
119	166	120	c183d82401800cfd0e14089803
120	157	120	c03809a3a138008d0301f41031
121	71	119	cf0412f2215600cc3180c8312080
122	169	119	c59f350010b12680bc4c00c066

more-ret jump-j quit/select-q;  
part list, quit-q select-s:s  
enter rule index 1-10637:345

listing from index 99, and selecting rule  
345 (not necessarily in the list)

Figure 33.17: Listing the sorted rule sample in figure 33.15 with 10637 *v4k7* kcodes. *Columns from left to right*: rule index number, mean entropy, variability (min-max or sdev), and the rule in hex — as much as will fit. The list order is firstly by decreasing variability *v*, then by decreasing mean entropy within each measure of *v*. Sets of rules separated by horizontal lines have the same *v* (max 0.18 because any *v*>0.18 is reset to 0.18). Various options are presented below the (partial) list: **m** to see more, **j** to jump to an index, **s** to select a rule.

If **l** for a full sample, or **p** for a patch (if active), is entered in section 33.5, rules in a sample or patch are presented in a top-right window as in figure 33.17. In a sorted list, rules are ordered by decreasing variability (min-max or sdev), then by decreasing mean entropy within each measure of variability, otherwise the list is in the order saved.

The title in the list window (e.g. **10637 v4k7 rules mean minmax**) shows the number of [*v*, *k*] rules in the sample. The rules are then listed, as many as will fit in the window, with the rule index on the left, followed by the mean entropy and variability (min-max or sdev) presented according to coordinates 0-255, and the rule in hex — as much as will fit. Sets of rules separated by horizontal lines have the same variability. The following prompt is show at the foot of the list,

more-ret jump-j quit/select-q:



*options ... what they mean*

**more-ret** ... if there are more rules in the sample, enter **m** to see the next batch.

**jump-j** ... to jump to a new rule index. The following extra prompt is presented,

**enter rule index 1-10637:** (*for example*)

Enter the rule index number which becomes the first in the new list.

**quit/select-q** ... if **q** is entered, the following extra prompt is presented,

**part list, quit-q select-s:**

Enter **q** to backtrack, or **s** to select a rule and see its space-time patterns (section 33.8.1 below).

### 33.8.1 Selecting a rule from the list

Enter **s** in section 33.8 or 33.9 to select a rule from the list. The following extra prompt is presented at the foot of the list window,

**enter rule index 1-10637:** (*for example*)

Enter any rule index number within the range. The program reverts to the prompt in section 33.5. Enter **q** (backtrack to the interrupt prompt — section 32.14), then enter **return** to show the space-time patterns of the selected rule.

## 33.9 Listing by plot coordinates or probing with the mouse

Entering **c** in section 33.5 or clicking the mouse within the scatter plot in section 33.6.1 allows the rule or rules at and adjacent to the chosen coordinates to be listed.

If **c** is entered, the following top-right prompts are presented in sequence,

**mean entropy pos 0-255:    min-max pos 0-255:** (*or sdev*)

Enter the coordinates to list the rule/s at the grid reference, a number (0 to 255)<sup>6</sup> in each case. The same can be achieved by clicking on the plot with the left mouse button (section 33.6.1).

The list will appear as in figure 33.18, similar to figure 33.17, but with a slightly different title, and options at the foot of the list, for example,

*title*

**10637 v4k7 rules y=143 x=108 rad=1** (*from figure 33.18*)

**left mouse button to select** (*reminder to set coordinates with a mouse click*)

This shows the coordinates selected, and the current radius, initially 1 unit (**rad=1**), resulting in any rule within a 3×3 block centered on the coordinates to be listed. A small radius may not contain any rules. The options at the foot of the list allow the radius to be expanded up to 9 units to include more rules — maximum block 19×19. The options differ between a short complete list and a long partial list, for example,

<sup>6</sup>The  $y, x$  coordinates are plotted according to unsigned char values, so the mean entropy (0 to 1) and the variability (0 to 0.18) are coarse-grained to the range 0 to 255. Any variability above 0.18 is reset to 0.18.

```

10637 v4k7 rules y=143 x=108 rad=1
left mouse button to select
155 143 109 cc6c0030020c3301307b6c4940

quit-q select-s rad-(max 9):
rad=1

10637 v4k7 rules y=143 x=108 rad=5
left mouse button to select
155 143 109 cc6c0030020c3301307b6c4940
165 148 106 f92ac00d0388d01c204002a1d1
173 144 105 c4082005da74f44403f4403513
180 147 103 f2408b080cc03463180302385d

quit-q select-s rad-(max 9):■
rad=5

10637 v4k7 rules y=143 x=108 rad=9
left mouse button to select
134 146 115 e00c0e4f01803310910011ec01
135 138 115 fc010219a014114b260d380030
141 142 114 c0038e0019036e0dc50c041db0
144 137 112 e43db0ac30302164308b0010f4
149 137 111 f0b2b00c08c00412414568549
151 149 110 f2030000e1641247513ee03a44
154 150 109 f2dca3c830430f39180b403c1a
155 143 109 cc6c0030020c3301307b6c4940
165 148 106 f92ac00d0388d01c204002a1d1
173 144 105 c4082005da74f44403f4403513
176 137 104 ca0cc90110213b030020e0e840
180 147 103 f2408b080cc03463180302385d

quit-q select-s rad-(max 9):■
rad=9

```

Figure 33.18: Listing the sorted rule sample in figure 33.15 by plot coordinates or probing with the mouse. The layout is the same as in figure 33.17, The coordinates ( $y=143$   $x=108$ ) were selected by clicking with the mouse (or by entering the values) with the default radius=1. The radius was expanded to 5, then to 9 to capture more rules on the scatter plot. *Columns from left to right:* rule index number, mean entropy, variability (min-max or sdev), and the rule in hex — as much as will fit. Various option are presented below the list: **q** to backtrack, **s** to select a rule (section 33.8.1), or **rad-(max 9)** to change the radius. A new mouse-click or **return** restores the scatter plot prompt (section 33.6) ready for a new mouse-click.

*options for a short complete list*

**quit-q select-s rad-(max 9):**

*options for a long partial list, presented in turn*

**more-ret jump-j quit/select-q:** (then if **q** is selected)

**quit-q select-s rad-(max 9):**

Enter a number up to 9 to expand the radius. Other options are the same as in section 33.8.

## 33.10 Rule sample encoding

A rule sample file (`.sta`) is a binary file of an unsigned char array, recording (A) each rule, (B) its mean entropy, and (C) the entropy variability (min-max or sdev). The file records A,B,C for each rule in the sample. In a sorted list, rules are ordered by decreasing variability, then by decreasing mean entropy within each measure of variability, otherwise the list is in the order saved.

The encoding of A,B,C is as follows: A (the rule-table) consists of bits as described in section 16.16, but omitting the two leading bytes for  $v$  and  $k$ . B and C are both whole numbers (0 to 255) and are stored as one byte each. The file does not record  $[v, k]$  so loading a file that was created with conflicting  $[v, k]$  will result in a meaningless scatter plot.

# Chapter 34

## Learning, forgetting, and highlighting

*not in TFO-mode.*

Attractors classify state-space into broad categories, the network’s “content addressable” memory in the sense of Hopfield [17]. 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 [32, 33].

DDLab provides tools for “sculpting” the basin of attraction field towards desired categories and sub-categories by methods that automatically attach/detach one or more “aspiring pre-images” to/from a “target-state”. The results and side affects of learning can be seen most clearly for a basin of attraction field, but the methods (allowing larger networks) also apply to a single basin, a subtree, or simply to a space-time pattern running forward.

---

### 34.1 Learning/forgetting methods

The learning methods apply to rcode only (i.e. not in TFO-mode), and work by adapting network architecture — mutating rules and/or moving wires. 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 or forgetting.

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 — those not on the list may also be learnt as a side effect. There is just one specific rule scheme mutation, which is bound to succeed. By contrast, there are many alternative wire moves that may achieve the desired result, though success is not guaranteed, and pre-existing pre-images may be forgotten [32].

Any change, especially in a small network, usually has significant side effects<sup>1</sup>, but this can be beneficial because of generalization, were a subtree is transplanted together with the attached pre-image as in figure 34.2. The revised attractor basin can be immediately re-drawn to show the results and side affects of learning. The algorithms for forgetting pre-images cause less severe side effects because forgetting requires only minimal changes to network architecture.

---

<sup>1</sup>These learning methods are not good at creating categories from scratch, i.e. solving the inverse problem or reverse engineering. A rudimentary solution to that is provided in section 18.7.4. However, the learning/forgetting methods are useful for fine tuning a network which is already close to where its supposed to be.

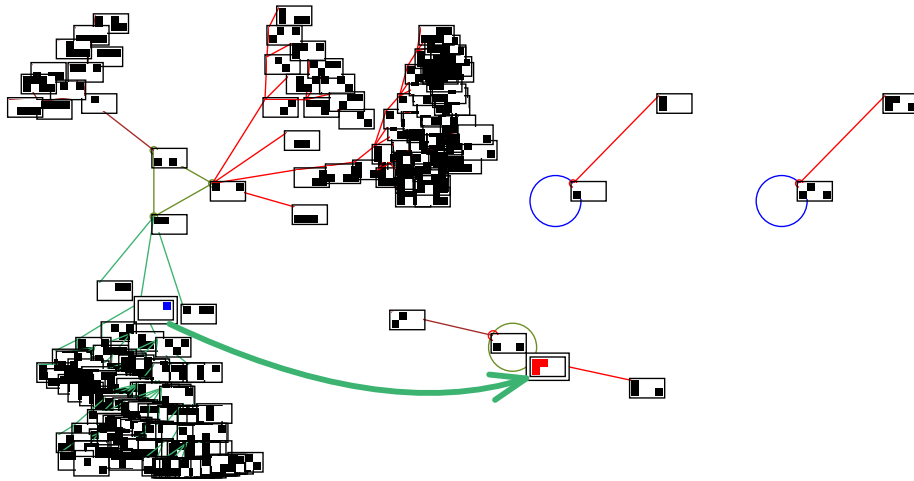


Figure 34.1: The basin of attraction field of a RBN,  $v2k4$ ,  $n=8$ , before learning. The red (arrowed) state has been selected as the target, and the blue state  $\square$  (at the tail of the arrow) as the aspiring pre-image with “highlight only”. Both have a double outline. The difference between the actual successor state  $\square$  and the target state  $\square$  is one bit (element 3), which should minimize side effects. Note that the basin layout was rearranged within the jump-graph, section 20.7.

The wiring algorithm requires nonlocal wiring. 2d or 3d CA are always treated as nonlocal, but 1d local wiring needs to be redefined as nonlocal in the learning prompts (section 34.5) or preset as nonlocal (section 12.4.1). The rule algorithm requires a rulemix set in chapter 14 — note that a single rule can be set up as a rulemix (section 14.4.3).

### 34.1.1 Highlighting states in attractor basins

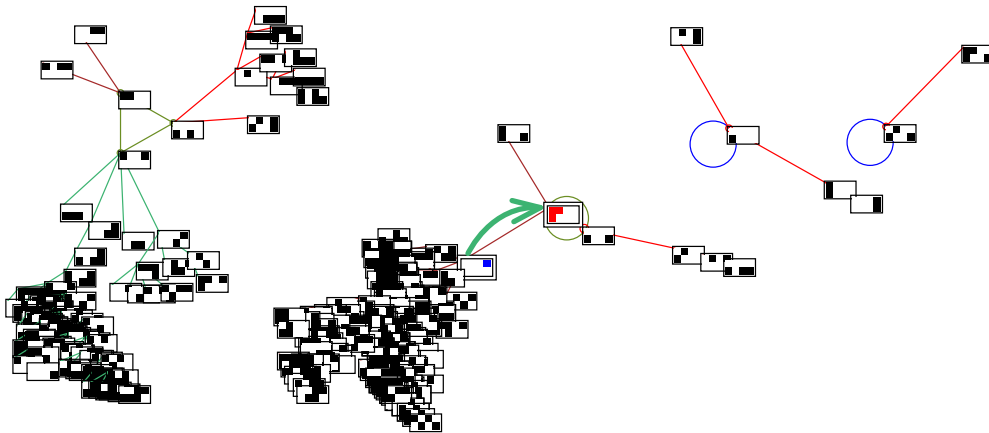
The nodes corresponding to a list of states can be highlighted in attractor basins — highlighting has no effect for space-time patterns. Highlighting shows up the results of learning and forgetting, but is also useful (without learning) to see how a particular set of states is distributed in the basin of attraction field, single basin or subtree, and how the distribution changes when network parameters are altered. The highlighted presentation is illustrated in figure 34.4 for different types of node display<sup>2</sup>.

For binary ( $v=2$ ) networks, 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. For  $v \geq 3$  networks, highlighted nodes shown as bit patterns correspond to values. Nodes not highlighted may be suppressed entirely. The highlighting feature may be used for any network architecture, including local 1d CA with compression on (section 26.2).

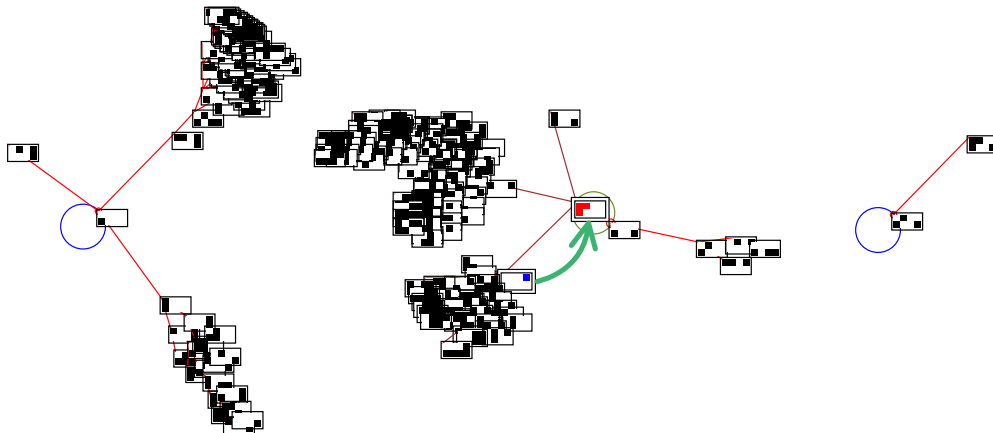
### 34.1.2 Basin layout for learning

The default layout of the basin of attraction field may need some adjustment for a good presentation for learning. Setting the layout is described in section 25.2, but can be readjusted with much greater

<sup>2</sup>The node display is selected in section 26.3 and is not affected by the presentation for selecting the target and pre-image states in sections 34.3 and 34.4.



(a) 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.



(b) The result of one example of learning by wire moves, with greater side effects. There are multiple possible solutions, but there may also be no solution by wire moves.

Figure 34.2: The basin of attraction field of a RBN,  $v2k4$ ,  $n=8$ , after learning (figure 34.1). (a) By bit-flips. (b) By wire moves. The target state and its pre-image are highlighted with a double outline. Note that the order and orientation of the basins has changed.

	<b>3. 2. 1. 0.</b>	<b>r</b> code(hex)
<b>7:</b>	3 1 6 0	<b>d2 33</b>
<b>6:</b>	1 0 4 4	<b>50 3a</b>
<b>5:</b>	5 0 0 2	<b>f6 b4</b>
<b>4:</b>	1 1 2 5	<b>7d 74</b>
<b>3:</b>	5 1 7 4	<b>b0 c5</b>
<b>2:</b>	3 0 0 5	<b>06 00</b>
<b>1:</b>	4 2 3 3	<b>9c 67</b>
<b>0:</b>	3 6 5 4	<b>12 00</b>

Figure 34.3: The RBN network for figure 34.1 before learning. Because the difference between the actual successor state and the target state is confined to element 3, learning changes effect this cell only. The new rule at element 3 (figure 34.2(a) is rcode(hex) b0c7 — one bitflip. The new wiring at element 3 (figure 34.2(b) is 5 1 7 2, one wire move.

flexibility in the jump-graph (section 20.7) – this was done for figure 34.1.

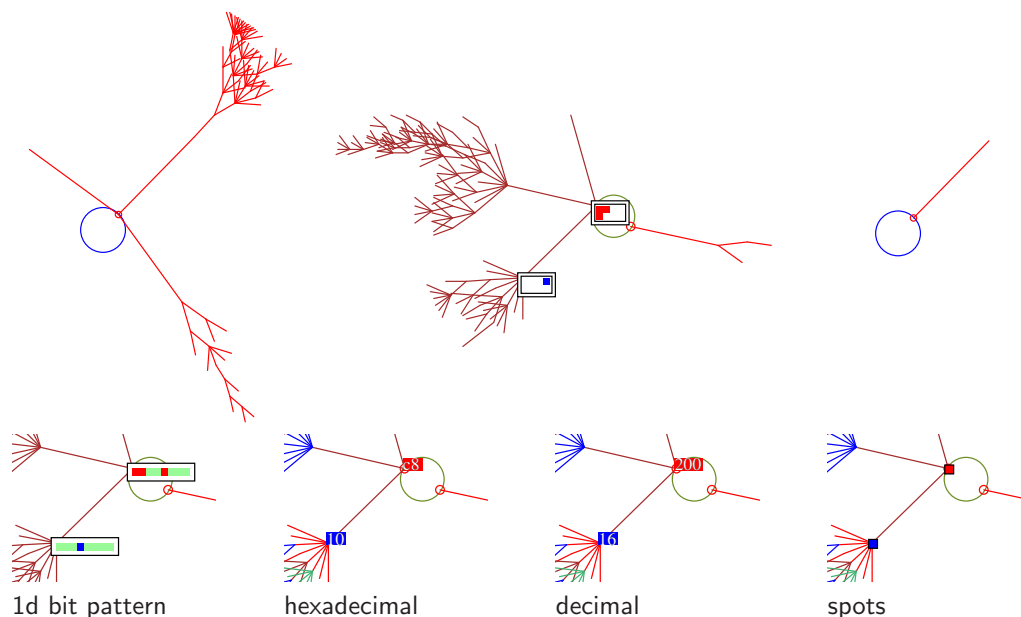


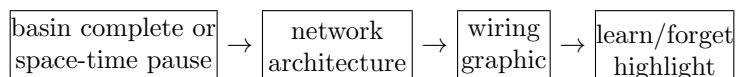
Figure 34.4: *Top*: the basin of attraction field of the RBN in figure 34.2(b),  $v2k4$ ,  $n=8$ , learning by wire moves. The target and pre-image nodes (states) are shown as  $2 \times 4$  bit patterns and highlighted with a double outline — other nodes are suppressed. *Below*: a fragment of the second basin showing alternative presentations of highlighted nodes (states). The presentation will follow the node display selected in section 26.3 irrespective of the presentation for selecting the target and pre-image states in sections 34.3 and 34.4, but other nodes can be suppressed if required.

## 34.2 Selecting the learn/forget/highlight window

Once rule/s have been set, whenever the network is reviewed in the lower-left wiring graphic (section 17.3), the learn/forget/highlight window and functions (section 34.3) can be activated. The upper-right wiring graphic reminder (section 17.4) includes the following option (highlighting does not apply for space-time patterns),

... **hilight/Lrn-l/L**      *or if interrupting space-time patterns ... Learn-L*

Enter **L** for the learn/forget/highlight functions. For attractor basins, enter **l** to highlight selected states. Sections 34.2.1 and 34.2.2 describe how to get to this point in DDLab — the sequence of prompt windows is summarized as below,



Note that changes to wiring and rules can be made directly from the wiring graphic, which can be done in conjunction with automatic learning/forgetting.

### 34.2.1 Selecting the wiring graphic

The wiring graphic (section 17.3) can be reached from the top-right network architecture prompt (section 17.1), for example,

```
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:
```

Enter **1**, **c**, **2** (or **3** for 3d) for different presentations of the wiring graphic (e.g. figures 17.2, 17.3, 17.10, 17.15).

### 34.2.2 Selecting the network architecture prompt

The network architecture prompt (section 17.1) can be reached as follows,

initial prompts ... during the main prompt sequence after rules are set — the network architecture prompt appears automatically (section 17.1).

attractor basins ... when a basin of attraction field, single basin or subtree is completed or abandoned, the following top-left “Attractor basin complete prompt” appears (section 30.4),

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

enter *net-n*,  
alternatively *ops-o* followed by *net-n* (section 30.5),

space-time patterns ... when interrupting space-time patterns, the top-right “Space-time pattern interrupt/pause prompt” (section 32.16) appears, for example,

```
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
on-the-fly-y hide-W no-ops-Q back-q cont-ret:
```

enter *net-n* for the network architecture prompt.

## 34.3 Select the target state

Enter **L**, or **l** (for highlighting only) in section 34.2 for the “target state” prompts in a lower left window, similar to “The seed prompt” (section 21.1) — the various methods of setting states are described in chapter 21.

The initial prompts sets the “target state” — to which “aspiring pre-images” will attempt to directly attach or detach themselves by means of learning/forgetting algorithms. Prompts are presented in two separate windows, one above the other. The upper window indicates that the target state is to be set,

*for attractor basins, **L** or **l** selected in section 34.2*

**learn/forget/highlight: set target state**    *or*    **highlight only: set target state**

*for space-time patterns, **L** selected in section 34.2, where highlighting does not apply*

**learn/forget: set target state**

The lower window presents the first of a series of prompts to select the target state, for example,

**Select target state (v2 1d n=14), win-w empty-e fill-f prtx-x** (*for 1d, v=2*)  
**rnd-r bits1d-b bits2d-B hex-h dec-c repeat-p load-l (def-B):**

Select the target state as if selecting a seed in section 21.1 — the prompts are the same, but the default selection method is initially **bits2d-B** (sections 21.4, 21.4.7).

## 34.4 Select aspiring pre-image/s

Once the target state has been selected, one or more states are specified as its “aspiring pre-images” (i.e. direct predecessors). The learning/forgetting algorithms will attempt to directly attach (learn) these states, or detach (forget) existing pre-images of the target state. Alternatively the “aspiring pre-images” and the “target state” can be just highlighted. The set of aspiring pre-images can be an arbitrary list of states, a range of decimal equivalents (if applicable — section 21.6), or can be based on Hamming distance, parity, or the previous list may be repeated. Prompts are presented in two separate windows, one above the other — the upper prompt appears first as follows,

**pre-images: as before-b parity-p range-r Hamm-h1/h2 list-(def 1):**  
*(parity-p and Hamm-h2 for v=2 only)*

These options are summarized below, with further details in the sections indicated,

*options ... what they means*

**before-b** ... to repeat a previous set of aspiring pre-images (parity, range or Hamm).  
 If a list was previously selected, the new list defaults in section 34.4.4 will correspond to the previous list.

**parity-p** ... (*v=2*) to select states based on odd or even parity (section 34.4.1).

**range-r** ... to specify a range of states according to their decimal equivalents, but within the limits of max-*n* in section 21.6.

**Hamm-h1/h2** ... enter **h1** for states with Hamming distance 1 from the target state.  
 For *v=2*, **h2** gives a Hamming distance of both 1 and 2 (section 34.4.3).

**list-(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 (section 34.4.4).

Once set, the target state and the set of aspiring pre-images (however selected) are reviewed as decimal equivalents (section 34.4.5).



### 34.4.1 Odd or even parity

*binary  $v=2$  systems only*

In a binary  $v=2$  network, odd and even parity signifies that the total of 1s in a state's bitstring is odd or even<sup>3</sup> (a zero total of 1s has even parity). By default, parity applies to the whole bitstring (size  $n$ ) if  $n \leq 20$ , otherwise only the 20 leading bits are considered. A smaller number of leading bits can also be specified.

Enter **p** in section 34.4 to automatically select aspiring pre-images with the given parity.

The following upper prompt is presented,

**parity: odd-1, or even-(def):**

Enter **return** for even parity, **1** for odd parity. A further upper prompt allows the parity to relate to a leading part only of the network,

**enter part system size (def 10):** (*for  $n=10$* )

If a size  $x \leq n$  (max  $x=20$ ) is entered, states with the given parity relating only to cell indexes 0 to  $x-1$  will be selected. Figure 34.5 shows the results of entering 7 in a  $n=10$  network.

```
enter part system size (def 10):7
target=0, pre-images=64, parity=even, max state=126, cont-ret:█
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 34.5: Selecting even parity for the leading 7 bits in a 10 bit network. A list of the  $2^7/2=64$  states with even parity are displayed in decimal. This is one kind of review list covered in section 34.4.5.

### 34.4.2 Range of decimal equivalentents

Enter **r** to automatically set a range of aspiring pre-images, set between two selected decimal values within the limits max- $n$  in section 21.6. For a basin of attraction field,  $n$  will be within these limits, but if  $n \geq \text{max-}n$  (likely for space-time patterns), so outside the range of allowed decimal equivalentents, only max- $n$  leading bits/values are considered. The following (upper) sequence of prompts is presented to set the start and end decimal equivalentents, and the “step” or increment size (default 1),

**range of pre-images (def 1-8, max 255), start: end: step:** (*for  $n=8$* )

### 34.4.3 Pre-images according to Hamming distance

Enter **h1** in section 34.4 to automatically set the aspiring pre-images based on a Hamming distance of 1, where states differ by just 1 bit/value from the target state. For  $v=2$  only, enter **h2** for a Hamming distance of both 1 and 2.

<sup>3</sup>The parity does not relate to the state expressed as a decimal value, but simply to the total of 1s irrespective of their position in the bitstring.

### 34.4.4 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, at the (upper) prompt (section 34.4),

... **list-(def 1):**

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 (section 21.1), except for the heading,

**Select aspiring pre-image 1 ...**

for a list, this is followed by ...

**Select aspiring pre-image 2 ...**

**Select aspiring pre-image 3 ... etc.**

The default selection method is setting 2d bits (sections 21.4 and 21.4.7).

### 34.4.5 Review target state and aspiring pre-images

Once the target state (section 34.3) and aspiring pre-images (section 34.4) have been set, information about the selection will be displayed in the lower window. If the system size is within the limits for decimal,  $n \leq \text{max-}n$  (section 21.6), the states in decimal will also be included. Figure 34.5 gives an example for parity. Examples for a range of decimal equivalents, Hamming distance, and an arbitrary list, are given below for  $v=2$ ,  $n=8$ ,

*a range of decimal equivalents, section 34.4.2*

**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,**

*the set of 1-bit mutants, Hamming distance=1, section 34.4.3*

**target=240, pre-images=8, 1-bit mutants, step=16, cont-ret:  
241,242,244,248,224,208,176,112,**

*an arbitrary list of 5 states, section 34.4.4*

**target=240, pre-images=5, cont-ret:  
50,167,152,88,94**

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:**

Enter **q** to backtrack to section 34.4 and reset the aspiring pre-images.

## 34.5 Learn, forget, or highlight only

Once the target state (section 34.3) and aspiring pre-image/s (section 34.4.4) have been set, reviewed and accepted (section 34.4.5), prompts are presented to learn, forget, or just highlight the aspiring pre-images (as well as the target state). Note that highlighting does not apply for space-time patterns.

### 34.5.1 1d CA — local wiring and highlighting

For a 1d CA (with local wiring and a 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, if L was selected in section 34.2 (if l was selected, highlight only applies)*

**local 1d: nonlocal wiring-w:**

*1d local wiring and mixed rules*

**local 1d, rulemix: nonlocal wiring-w:**

Enter **w** to redefine the wiring as nonlocal, allowing learning by wire moves, and suppressing compression in attractor basins (section 26.2). If **return** is entered to conserve local wiring, subsequent options differ for space-time patterns or attractor basins. For space-time patterns, the program skips directly to the last prompt before exiting the learning routine (section 34.9).

For attractor basins, there is a reminder that although learning is deactivated, the highlighting options remain,

*for attractor basins*

**... highlight only-(def):**

Enter **return** to skip directly to the highlighting options (section 34.8). Compression in attractor basins is suppressed (section 26.2).

Highlighting is the only possibility for a network were the wiring is defined as local 1d, using the local 1d reverse algorithm (section 2.19). Note that 1d local wiring can also be redefined so that it is 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.

### 34.5.2 Nonlocal wiring — learn, forget, or highlight only

For networks that do not have 1d local wiring, or if **w** was entered in section 34.5.1 above, or for any other network with nonlocal wiring, including 2d and 3d CA<sup>4</sup>, the following prompt is presented,

*for attractor basins*

**highlight only-h, forget-f, learn-(def):**

*for space-time patterns, where highlighting does not apply*

**forget-f, learn-(def):**

Enter **return** or **f** to learn or forget i.e. add/remove the aspiring pre-image/s to/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 (section 34.1.1).

Enter **h** to just highlight the selected states (without learning or forgetting) — the program skips directly to the last prompt before exiting the learning routine (section 34.9).

---

<sup>4</sup>2d and 3d CA “local” wiring is treated as nonlocal in DDLab’s algorithms.

### 34.5.3 Learning/forgetting by wire moves or bit/value-flips

If learning or forgetting was selected in section 34.5.2, a prompt (in the upper window) is presented to proceed either by moving wires or flipping bits/values in rule-tables.

For a single rule network, only learning/forgetting by wire moves is possible<sup>5</sup>, and the prompt is as follows,

**learn by wire-moves only-(def):** (*or forget ...*)

Enter **return** to proceed by wire-moves (section 34.7).

Otherwise, for networks with nonlocal wiring and a rulemix, and also networks with mixed- $k$  which by definition has both, the prompt is as follows,

*for binary,  $v=2$ , networks*

**learn by wire-moves -w, bit-flips-(def):** (*or forget ...*)

*for multi-value,  $v \geq 3$ , networks*

**learn by wire-moves -w, value-flips-(def):** (*or forget ...*)

Enter **return** for bit/value-flips (section 34.6), or **w** for wire-moves (section 34.7).

Enter **q** to backtrack.

## 34.6 Learning/forgetting by bit/value-flips

The bit/value-flip learning algorithms [32] change a specific bit/value in the rule-table at each cell position where there is a mismatch between the actual successor cell of the aspiring pre-image and the same cell in the target state. This takes care of that specific mismatch for all future aspiring pre-images, so pre-existing pre-images cannot be “forgotten”, i.e. detached as pre-images from the target state, by learning more states as pre-images.

However other states not on the list of aspiring pre-images may also be learnt, so side effects will occur elsewhere in the attractor basin. To “forget” a pre-image, just one of a set of possible bit/value-flips is sufficient, chosen from the set at random. As fewer network changes are needed there will be fewer side effects.

If **bit-flips** or **value-flips** was selected in section 34.5.3, the required changes to the network will be made, and the following (upper) prompt is presented,

*for attractor basins*

**learn/forget/highlight more-m:**

*for space-time patterns, where highlighting does not apply*

**learn/forget more-m:**

Enter **m** to repeat the procedure from section 34.5, otherwise the network as it stands is accepted. For attractor basins the node highlighting options follow (section 34.8). For space-time patterns the program skips directly to the last prompt before exiting the learning routine (section 34.9).

<sup>5</sup>section 14.4.3 describes how to set up a single rule network that is treated as a rulemix.



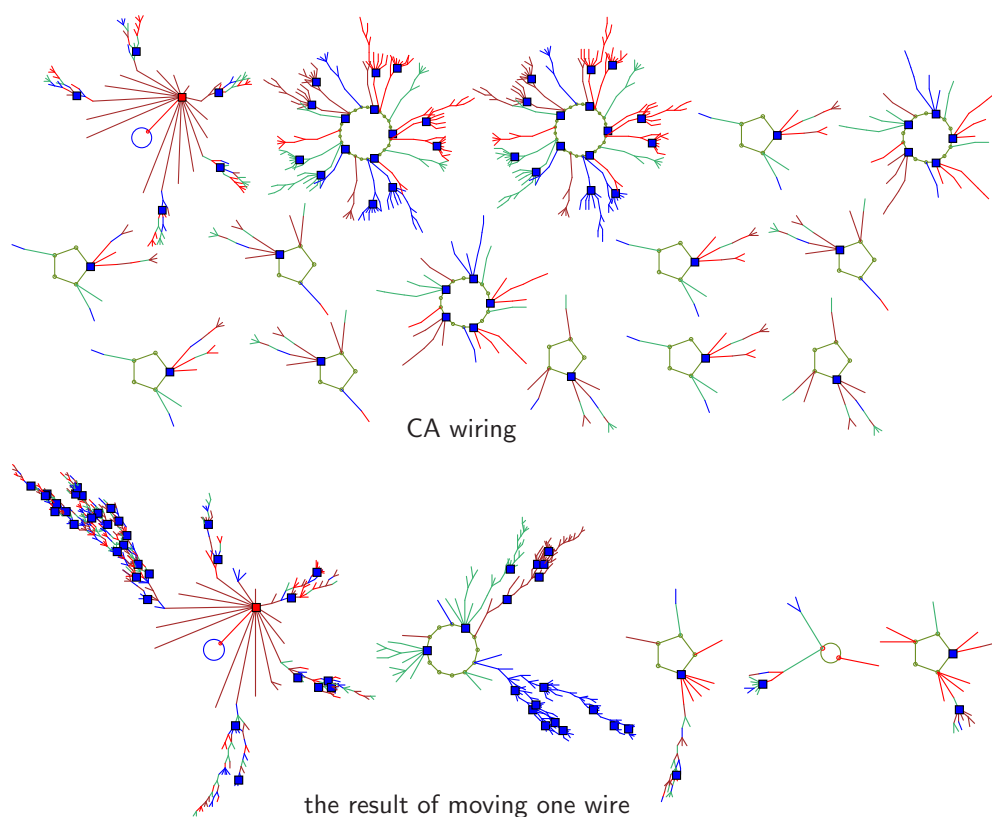


Figure 34.6: *Top*: the basin of attraction field of the CA  $v2k3$  rcode 110,  $n=10$  (compression suppressed). *Bottom*: the result of moving one wire, cell 4 wiring 5 4 3 changed to 5 9 3, done by “hand rewiring” in section 17.9.4. The node type selected was `spot` in section 26.3. In both fields, the target state (all-1s) is highlighted in red, together with states Hamming distance 1 and 2 from the target state (`h2` section 34.4) highlighted in blue. All other states are suppressed. Note that the basin layout was rearranged within the jump-graph, section 20.7.

If `w` is selected, the last `<-no` state will be learnt 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 (upper) prompt is presented,

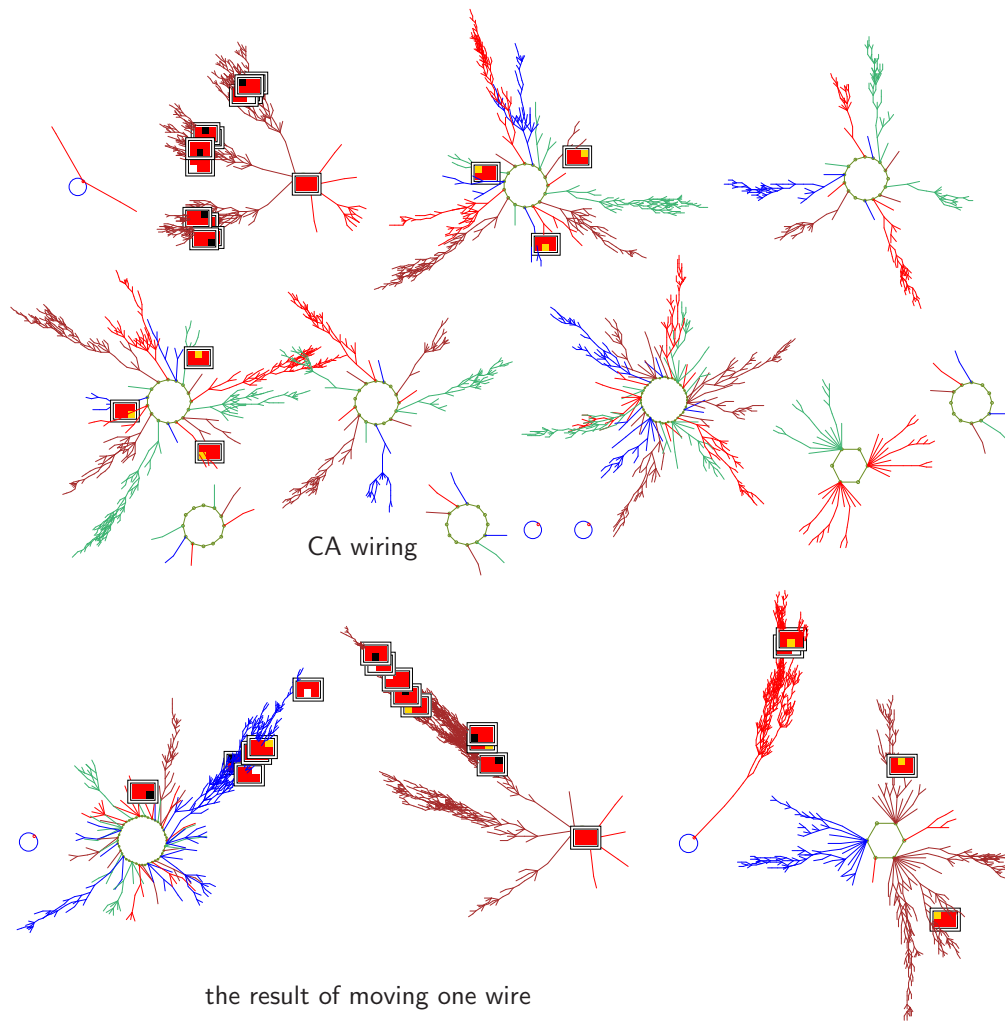


Figure 34.7: *Top*: The basin of attraction field of a CA  $v4k3$ ,  $n=6$  (compression suppressed). *Bottom*: the result of moving one wire, cell 2 wiring 3 2 1 changed to 3 5 1, done by “hand rewiring” in section 17.9.4. The node type selected was **2d-B** in section 26.3. The highlighted states are the all-red target (all-2s), together with states Hamming distance 1 from the target state (**h1** section 34.4). All other states are suppressed. Note that the basin layout was rearranged within the jump-graph, section 20.7. (rcode(hex)96570e1b37b0b8d6b3189e967bac51ac).

**learn/forget/highlight more-m:** *or for space-time patterns learn/forget more-m:*

Enter **m** to repeat the procedure from section 34.5, otherwise the network as it stands is accepted. For attractor basins the node highlighting options follow (section 34.8). For space-time patterns the program skips directly to the last prompt before exiting the learning routine (section 34.9).

## 34.8 Highlighting options

*attractor basins only*

Once learning is complete, or if *highlight only* was selected in section 34.5, the nodes representing the target state and the set of aspiring pre-images can be highlighted in the attractor basin — the particular form of highlighting depends on the node type selected in section 26.3.

The following prompt is presented,

**highlight: none-n, suppress all other nodes-s:**

Enter **s** to suppress the display of nodes other than highlighted nodes. Enter **return** to highlight the selected nodes and show other nodes normally according to the selected node type. Enter **n** not to highlight, but show all nodes as normal. Examples are shown in figures 34.4, 34.6 and 34.7.

---

## 34.9 Learning/forgetting/highlighting complete

When learning/forgetting or highlighting is complete, the lower-left wiring graphic (section 17.3) and the “wiring graphic reminder” (section 17.4) reappear. Within the wiring graphic, the results of learning/forgetting can be re-examined and changes made, for example with “hand rewiring” (section 17.9.4) or “revising the rule” section 17.9.10.

Enter **return** to exit the wiring graphic, and revert to the top-left “attractor basin complete” prompt (section 30.4), or the pause options for space-time patterns (section 32.16).

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 shows how a set of highlighted states is redistributed for a succession of mutated attractor basins (figures 34.6, 34.7). However, when “sculpting” the basin of attraction field, it is safer to first deactivate mutation in section 28.1.

---



# Chapter 35

## Filing

DDLab has functions for saving and loading a variety of DDLab specific file types. This chapter lists the file types, and describes the methods for saving and loading files. The file types are described in more detail in other sections of the manual as noted.

---

### 35.1 File naming constraints in DDLab

DDLab was first developed in DOS, and DOS filename conventions still apply for all versions, including Linux-like operating systems. A DOS filename has up to eight characters (starting with a letter) and up to three characters in extension following the dot. DDLab file names have a three letter extensions which identify the file type, for example `myru1_v3.ru1`, except for PostScript files generated in DDLab have a two letter extension, for example `my_3dPS.ps`. The extension is not entered as part of the filename when saving or loading — it will be automatically added according to the file type.

The lowercase letter `q` cannot be included in a file name because it is used for backtracking or deleting input from the keyboard.

---

### 35.2 File types, default filenames, and extensions

The DDLab specific file types, default extensions, and default filenames are listed below.

<i>type</i>	<i>default filename</i>	<i>description</i>
<b>K-MIX</b>	<code>my_mix.mix</code>	The neighborhood-mix, (sections <a href="#">9.4</a> , <a href="#">9.11.3</a> , <a href="#">19.5</a> ). Encoding: see section <a href="#">19.3.1</a> .
<b>WIRING ONLY</b>	<code>mywso.w_s</code>	The wiring scheme (chapter <a href="#">19</a> ). Encoding: sections <a href="#">19.3</a> .

<b>SINGLE RCODE</b>	<code>myrul_v2.rul</code>	... for rcode ( <i>for example</i> )
<b>SINGLE KCODE</b>	<code>myvco_v3.vco</code>	... for kcode ( <i>for example</i> )
<b>SINGLE TCODE</b>	<code>mytco_v4.tco</code>	... tcode ( <i>for example</i> )
	etc.	A single rule, rcode, kcode or tcode, for a given $v$ and $k$ (chapter 16, and sections 30.5.1, 32.16.1). Encoding: section 16.16. The default filename includes $v$ . When setting a rule, kcode, or tcode, in chapter 16, the most recent will be automatically saved, with the filename <code>l_v2k3.rul</code> , <code>l_v2k3.vco</code> , <code>l_v2k3.tco</code> — including both $v$ and $k$ — these rules can then be automatically reloaded.
<b>RCODE</b> as 1d <b>SEED</b>	<code>mysee_v2.eed</code> <code>mysee_v3.eed</code> etc.	The rule-table saved as a 1d seed (section 16.4.4). Encoding: section 21.9. The default filename includes the value-range $v$ . This is the same as a 1d <b>SEED</b> — below.
<b>RULE TABLE</b>	<code>mytbl_v2.tbl</code> <code>mytbl_v3.tbl</code> etc.	The rcode-table, or in TFO-mode the kcode-table or tcode-table, for a given value-range $v$ , can be saved or loaded as an ASCII file (section 16.22. The default filename depending on $v$ .
<b>RCODE SCHEME</b>	<code>myrso_v2.r_s</code>	... for rcode ( <i>for example</i> )
<b>KCODE SCHEME</b>	<code>myrso_v3.r_v</code>	... for kcode ( <i>for example</i> )
<b>TCODE SCHEME</b>	<code>myrso_v4.r_t</code> etc.	... for tcode ( <i>for example</i> ) The rcode, kcode, or tcode, scheme (with corresponding extensions <code>.r_s</code> , <code>.r_v</code> , <code>.r_t</code> (chapter 19). For homogeneous- $k$ only, otherwise use the WIRING+RULE SCHEME below. The default filename includes the value-range $v$ . Encoding: sections 19.3.
<b>WIRING+RULE</b> <b>SCHEME</b>	<code>mywrs_v2.wrs</code> <code>mywrs_v3.wrv</code> <code>mywrs_v4.wrt</code> etc.	... for rcode ( <i>for example</i> ) ... for kcode ( <i>for example</i> ) ... for tcode ( <i>for example</i> ) Both wiring and rules (chapter 19). Encoding: sections 19.3. The default filename includes the value-range $v$ .
<b>NETWORK DATA</b>	<code>my_net.dat</code>	... for the network data as an ASCII file (section 19.6).
<b>SEED</b>	<code>mysee_v2.eed</code> <code>mysee_v3.eed</code> <code>mysee_v4.eed</code> etc.	The seed or state, which includes the value-range $v$ , size $n$ , dimension (1d, 2d or 3d), and the $i, j, h$ coordinates (sections 21.4.2, 21.4.8, 30.5.2, 32.16.4). Encoding: section 21.9. The default filename includes the value-range $v$ . Note that the same filenames also apply for saving/loading a patch — part of a state (sections 21.4.1, 21.4.8).

<b>SEED (ascii)</b>	<code>Aseed_v2.see</code> <code>Aseed_v3.see</code> <code>Aseed_v4.see</code> etc.	The seed or state as an ASCII string (section 21.10), which includes the value-range $v$ , the dimension is not recorded. Selection: section 21.1. Description and encoding: section 21.10. The default filename includes the value-range $v$ .
<b>Golly ascii seed</b>	<code>Golly_v2.eed</code>	For 2d binary ( $v=2$ ) networks, the seed or state as an ASCII string following the Golly file format. Selection: section 21.1. Description and encoding: section 21.11).
<b>BASIN DATA</b>	<code>my_data.dat</code>	An ASCII file for the field/basin/subtree and list of states data (sections 27.3—27.5.2, 19.6). Encoding: ASCII.
<b>PRE-HIST-DATA</b>	<code>my_prh.prh</code>	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 onwards.
<b>HIST-DATA</b>	<code>my_his.his</code>	Data for various other histograms (sections 9.11.1, 17.9.13, 31.6.2, 31.7.5, 31.8). Encoding: a binary file of an unsigned short array, recording the height of each histogram bar, from zero onwards.
<b>EXHAUSTIVE</b>	<code>myexh_v2.exh</code> <code>myexh_v3.exh</code> <code>myexh_v4.exh</code> etc.	An exhaustive mapping, listing each state and its successor (section 29.8.1, 29.7.2, 18.1.2). Encoding: a binary file of a char array recording a list of $v^n$ successor bit/value strings, each in $v^n/8$ bytes (minimum 1 byte), where the pre-image is the array index, from zero onwards. The default filename includes the value-range $v$ .
<b>ORDER-DATA</b>	<code>my_order.ord</code>	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 onwards.
<b>STAT-SAMPLE</b>	<code>mystat.sta</code>	The statistical sample of automatically classified rule-space, (chapter 33, sections 33.4, 33.5, 32.16.2). Encoding: section 33.10.
<b>GRAPH LAYOUT</b>	<code>my_grh.grh</code>	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 $x, y$ coordinates of each graph node according to the array index, from one onwards.

**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`        ... space-time patterns (and rule-tables).  
`my_3dPS.ps`        ... 3d space-time patterns.  
`my_sgPS.ps`        ... space-time patterns within the network-graph.  
`my_ngPS.ps`        ... network-graph.  
`my_jgPS.ps`        ... jump-graphs.  
`my_bPS.ps`        ... attractor basins.  
`my_mxPS.ps`        ... wiring-matrix.  
`my_wgPS.ps`        ... wiring-graphic.

These are vector PostScript files described in chapter 36, which includes relevant sections for selection. The files can be loaded in GhostView (Linux command `gv &`), and included in LaTeX documents. Encoding: ASCII - in the PostScript language.

### 35.3 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 v=3 seed in Linux-like systems*

```
LOAD SINGLE RULE-filename (no ext) .rul will be added
change dir-d, now: /home/sussex/ddlabz04/ddfiles:
list-?, quit-q, default myrul_v3:
```

*saving a v=3 seed in DOS*

```
SAVE SEED-filename (no ext) .eed will be added
change dir-d, now: C:\WATCOM\ dddlabz04
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 loading and 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.

## 35.4 Loading constraints and warnings

When loading files for the  $k$ -mix, single rules, networks and sub-networks (rules, wiring, or both), and seeds (1d, 2d, 3d), top-right warnings, options and messages may be displayed if there appears to be a conflict or compatibility issue between the *file* and the *base* setup. This relates to compatibility with parameters such as value-range  $v$ , neighborhood  $k$  or  $k_{max}$ , and network size  $n$  or  $[i, j, h]$ . Details of the warnings, options and messages are shown in the relevant sections, as follows,

<i>file type ... loading compatibility sections</i>	
$k$ -mix file (*.mix) ...	9.4
single rules ...	16.14
single rules as ASCII string ...	16.22
networks and sub-network ...	19.4 — 19.4.4, 17.9.12
seeds ...	21.7 — 21.7.5
seeds as ASCII string ...	21.10
seeds in Golly format ...	21.11.1
exhaustive mapping ...	29.8.1
graph layout ...	20.9

## 35.5 Changing the directory

The current directory is shown in the filing prompt (section 35.3), and can be changed by entering *dir-d* in section 35.3— the following top-right prompt appears, for example,

*example for Linux-like systems*

**current path** /home/sussex/ddlabm07/ddfiles, **CHANGE dir**  
**new path or subdir:**

*example for DOS*

**current path** C:\WATCOM\DDLABM07, **CHANGE dir**  
**new path, (ie c:\dir\subdir):**

In Linux-like systems enter the sub-directory name, or “..” (dot dot) for the parent directory. In DOS, enter the full new path, (i.e. c:\newdir\newsub\mydir). Directory names are limited in the same way as filenames — 8 characters starting with a letter but excluding **q**.

The filing prompt (section 35.3) will reappear with the new path, which will stay current until changed again. If the new subdirectory or path is missing, or has an illegal name, the following error message appears,

**can't change to** [illegal name]

The program reverts to the filing prompt (section 35.3).

## 35.6 List files

To list the files with the current filename extension (in the current directory) enter **?** (question mark) in the filing prompt (section 35.3). A list will be displayed in a narrow window on the right of the DDLab screen (figure 35.1). The following prompt appears at the bottom of the list,

**list again-a** (*if the list fits on the screen*)  
**more-m** (*if there are too many files to fit*)  
**or filename**  
**:** (*the filename (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 35.3).

```

LOAD SINGLE K-CODE-filename (no ext) .vco will be added
change dir-d, now: /home/andy/sussex/ddlabm06/ddfiles:
list-?, quit-q, default myvco_v3:
v3k6x1.vco
v3k6x2.vco
v3k7w1.vco
v3k7w2.vco
l_v3k7.vco

```

Figure 35.1: Listing files with **?** (question mark) in section 35.3. Only the files with the current filename extension and in the current directory are listed.

## Chapter 36

# Vector PostScript capture of DDLab output

DDLab's graphics output can be captured as vector graphics in PostScript files, where images can be scaled without degrading their resolution — preferable for publication quality images over bitmap graphics. The methods work in both Linux-like systems and DOS. Images saved as vector PostScript `.ps` files can be viewed/printed in GhostView<sup>1</sup>, or converted to `.pdf` files and viewed/printed in Adobe (acroread). Most figures in this manual are vector PostScript graphics, others are bitmap graphics<sup>2</sup>.

Vector PostScript files are available for the following DDLab graphic output, listed below with the default filenames. Instructions for creating the files are given within the manual in the relevant sections indicated.

*default filename ... graphic output and relevant sections*

- `my_sPS.ps` ... For 1d or 2d snapshots including the 2d version of 3d.  
From seed information: section 21.4.10.  
By directly scanning space-time patterns: section 32.18.  
For the rule-table as a 1d seed: section 16.4.4.
- `my_3dPS.ps` ... For a 3d isometric snapshot from seed information: section 21.4.11).
- `my_sgPS.ps` ... For a snapshot of space-time patterns within the network-graph: section 32.19.2.
- `my_ngPS.ps` ... The network-graph: section 20.8.
- `my_jgPS.ps` ... The attractor jump-graph: section 20.7.1.
- `my_bPS.ps` ... Attractor basins of all types and presentations: section 24.2.  
For basins within jump-graph nodes: section 20.7.1.
- `my_mxPS.ps` ... The wiring-matrix: section 17.2.1.
- `my_wgPS.ps` ... The wiring-graphic (1d time-steps, 1d circle, 2d, 3d): section 17.9.14.

---

<sup>1</sup>GhostView (<http://pages.cs.wisc.edu/~ghost/>) — enter “gv &” in the terminal.

<sup>2</sup>Bitmap graphics can be captured with various screen grabbers such as XView (<http://www.trilon.com/xv/>) — enter “xv &” in the terminal. See also section 2.15.

A top right prompt for saving to PostScript image allows presentation alternatives, depending on the current presentation of the DDLab graphic, and with various options such as color or greyscale, and more options for space-time patterns or snapshots, such as divisions between cells. Then the top-right filing prompt (section 35.3) appears for naming the file which DDLab generates automatically, with the extension **.ps** added.

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 (section 2.15). 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.

The PostScript file cannot be viewed within DDLab, use GhostView.

## 36.1 Cropping and editing vector PostScript

Vector PostScript files are ASCII files, plain text files in the PostScript language, so can be easily edited, allowing the image to be cropped, images merged and annotations added. Instructions for the PostScript language can be found online<sup>3</sup>, so there is no need to delve further except to note how the image can be cropped, and the width of lines amended.

### 36.1.1 Cropping the PostScript image

The PostScript image can be cropped by editing the second line of the ASCII PostScript file to define a new BoundingBox with the bottom-left and top-right coordinates, for example,

```
%%BoundingBox: 20 40 924 670
```

The required coordinates can be found by a readout of the mouse pointer 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.

### 36.1.2 Amending the width of lines

The width of lines in various types of image generated by DDLab as vector PostScript (or converted to PDF) may look reasonable on the computer screen, but not in a high resolution printout, or vice versa. The defaults are a best compromise, but can be amended in the ASCII PostScript file by editing the value of “setlinewidth”, usually in the fourth line of the file — presently either .7 or .3, for the following file types according to the default filenames at the start of this chapter,

```
.7 setlinewidth ... my_ngPS.ps, my_jgPS.ps, my_bPS.ps,
                  my_mxPS.ps, my_wgPS.ps (for links and outlines)
.3 setlinewidth ... my_sPS.ps, my_3dPS.ps, my_sgPS.ps (for division lines and grid lines)
```

<sup>3</sup>For example, the PostScript Language Tutorial and Cook Book.



# Chapter 37

## 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.
- basic limits** ... where the size of the network  $n$  and neighborhood  $k$  have not been extended with the `exLimits` option in sections 1.6.1, 6.2.4.
- 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  $k$  inputs (1d, 2d, 3d) and one rule (but possibly a mix of rules to extend the definition).
- CA-wiring** ... same as **local wiring** below.
- cell index** ... (or network index) the way the network cells are numbered (sections 10.2.1 — 10.2.3), but cell index usually refers to a 1d network, or a 1d version of a 2d or 3d network.
- chain-rules** ... maximally chaotic rcode, where  $Z_{left} = 1$  or  $Z_{right} = 1$ , but not both, exhibit extremely low convergence in subtrees (section 16.11).
- 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.

- density-bias** ... the fraction of non-zero values randomly distributed in a rule-table (section 16.3.1) also known as the  $\lambda$ -parameter [22], or in a seed or block (section 21.3.2).
- ECA** ... Elementary Cellular Automata, binary, “nearest neighbor”  $k=3$  CA.
- effective  $k$**  ... can be smaller than actual  $k$  because of redundant wiring (section 18.7.3).
- elementary rules** ... 1d  $v2k3$  rcode make up the 256 elementary rules [28].
- equivalence classes** ... rules can be equivalent to each other by symmetry transformations, for example the 256 elementary rules collapse to 88 equivalence classes [31] (section 18.5). There will be more transformations for  $v \geq 3$  or greater dimensions.
- exhaustive algorithm** ...  
 ... finding pre-images by first creating 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).
- exLimits** ... an option allowing extended limits (as opposed to **basic**” **limits** for the size of the network  $n$  and neighborhood  $k$  (sections 1.6.1, 6.2.4). each state in state-space and its successor, for the exhaustive algorithm (section 29.7).
- field** ... as in basin of attraction field (chapter 23).
- FIELD-mode** ... to show the basin of attraction field, which does not require an initial state (the seed).
- full rule-table** ... rcode, where the rule-table (look-up table) lists the output for every possible neighborhood string.
- garden-of-Eden state** ...  
 ... a state having no pre-images, also called a leaf state.
- $G$ -density** ... the fraction of leaf states in a subtree, basin or field — a measure of convergence in network dynamics (section 24.9).
- hex** ... an abbreviation with two meanings according to the context: “hexagonal” referring to a 2d lattice or neighborhood, or “hexadecimal” referring to a numbering system for rules, as opposed to decimal.
- homogeneous- $k$**  ... 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  $\times$  depth, or columns  $\times$  rows.
- $I, J$**  ... the cell coordinates in a 2d  $i, j$  network.
- $i, j, h$**  ... the size of a 3d network, width  $\times$  depth  $\times$  height, or columns  $\times$  rows  $\times$  levels.
- $I, J, H$**  ... the cell coordinates in a 3d  $i, j, h$  network.
- input-entropy** ... the Shannon entropy of the input-frequency distribution in space-time patterns (section 31.5).

- input-frequency** ... the frequency of rule-table lookups, or neighborhood size blocks, in space-time patterns (section 31.5) usually presented as a histogram (e.g. figure 33.1).
- 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** ... homogeneous- $k$ , where each cell in the network has the same number of inputs, or neighborhood size,  $k$ . Note that a homogeneous- $k$  network can be created as a  $k$ -mix network if required.
- $k_{Lim}$  ... the maximum size of  $k$  (input wires to a cell) currently supported in DDLab, which depends on  $v$  and is more generous if TFO-mode is active (section 7.2, (table 7.1)).
- $k$ -mix** ... or mixed- $k$  heterogeneous- $k$ , 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.
- $k_{max}$  ... a deliberate selection of the maximum  $k$  in a  $k$ -mix. Note that  $k_{max} \leq k_{Lim}$ .
- $\lambda$ -parameter** ... for  $v=2$ , the fraction of 1s in the rcode-table. In general, the fraction of non-quiescent values [22] (section 24.9).
- $\lambda_{ratio}$  ... a normalized version [31] of the  $\lambda$ -parameter to compare directly with the  $Z$ -parameter. For  $v=2$ ,  $\lambda_{ratio}$  = the total of the lesser value ( $T_L$ ) (1s or 0s) divided by  $S/2$ , in general,  $\lambda_{ratio} = T_L / (S/v)$ .
- leaf state** ... a state having no pre-images, also called a garden-of-Eden state.
- Linux-like** ... Linux/MAC/CygwinX/UNIX/Irix versions of DDLab, as opposed to the DOS version.
- local wiring** ... or CA-wiring, where inputs may arrive from the local neighborhood as defined in chapter 10. Local wiring may be set up as nonlocal if required.
- lookup-table** ... a list of the outputs of possible neighborhoods, also known as the rule or rule-table (rcode, kcode or tcode) as defined in chapter 13.
- max- $k$**  ... same as  $k_{max}$  above.
- max- $n$**  ... maximum network size that can be expressed in decimal (section 21.6).
- max- $S$**  ... maximum rule-table size that can be expressed in decimal (table 16.2. Also applies to seeds (table 21.3).
- min-max** ... a measure of variability, the maximum up-slope — the rise in entropy following a minimum value (section 33.1 and figure 33.2) — the alternative is sdev — standard deviation.
- mixed- $k$**  ... same as  $k$ -mix above.
- $n$  ... the size of the network, or system size.
- $n_{Lim}$  ... the maximum size of the network. In FIELD-mode  $n_{Lim}$  varies according to the value-range  $v$  (Limits on network size for FIELD-mode,  $n_{Lim}$ ). 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).
- NBC** ... Null Boundary Conditions.
- 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 (predefined in chapter 10).
- network dimensions** ... 1d, 2d, or 3d, depending on the neighborhood (chapter 10).
- network-graph** ... a graphic representation of the network, which can be manipulated, showing all nodes and connections (chapter 20).
- network index** ... (or cell index) the way the network cells are numbered (sections 10.2.1 — 10.2.3).
- network size** ... the number of cells or elements in the network, also referred to as "system size".
- nonlocal wiring** ... where inputs may arrive from from anywhere, but may also be biased. Local wiring may be set up as nonlocal.
- null boundary conditions** ...  
 ... (NBC) where inputs beyond the network's edges are held at a constant value of zero, as opposed to periodic boundary conditions.
- outer-totalistic rules** ... where a different totalistic rule (tcode or kcode) applies according to the value of the center cell, so  $v$  rules. Applies to TFO-mode only, and not for mixed- $k$ . (section 13.7).
- partial order updating** ...  
 ... where a subset of cells updates synchronously, in parallel, followed by the next subset — then the "state" is the configuration after each updated subset (section 31.4.3).
- pattern density** ... the fractions of different values in space-time patterns (section 31.5).
- PBC** ... Periodic Boundary Conditions.
- periodic boundary conditions** ...  
 ... (PBC) where inputs wrap around to the opposite edge, as opposed to null boundary conditions.
- pseudo-neighborhood** ...  
 ... required for nonlocal wiring where inputs to a cell can arrive from anywhere. The nonlocal 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 10.1.1).
- pre-images** ... a state's immediate predecessors.
- RBN** ... Random Boolean Networks: random wiring of  $k$  inputs (but possibly with mixed- $k$ ) and a mix of rules (but possibly just one rule).
- rcode** ... the full rule-table (lookup-table) listing the outputs for every neighborhood string, with  $S = v^k$  entries.
- reaction-diffusion rules** ...  
 ... or excitable-media [14], 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).

- rotational symmetry (RS)** ... rotational equivalents — repeating segments (section 26.2).
- rule** ... any kind or rule or rule-table, including rcode, kcode or tcode, as defined in chapter 13.
  - rulemix** ... where each cell in the network can have a different rule (chapter 14).
  - rule-table** ... a list of the outputs of possible neighborhoods, also known as the lookup-table (rcode, kcode or tcode) as defined in chapter 13.
  - rule scheme** ... the actual rulemix (chapter 14).
  - sdev** ... the standard deviation of the entropy — a measure of variability (section 33.1 and figure 33.2) — the alternative is min-max.
  - 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.
- sequential updating** ... where each cell is updated in turn in some arbitrary order — then the “state” is the configuration when all  $n$  updates are complete (sections 29.9, 31.4.2).
- space-time patterns** ... the forward dynamics shown as a pattern for each successive time-step, which can be presented in many alternative formats described in chapters 31, 32.
- state** ... usually refers to the global state, the bit/value string, as opposed to the cell state or cell value/color.
- 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”).
- STP-dimensions** ... the dimensions of the space-time pattern presentation, which are not necessarily the underlying network dimensions.
- subtree** ... part of a transient tree in a basin of attraction defined by its root (chapter 23).
- synchronous updating** ...
- ... where all cells in the network update simultaneously, in parallel.
- system size** ... the number of cells or elements in the network, also referred to as “network size”.
- 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  $S = k(v - 1) + 1$  entries. (section 13.6.3).
- TFO-mode** ... stands for “totalistic forwards-only” and constrains DDLab to run just forwards to generate space-time patterns (section 6.2.1) — attractor basin functions are suppressed. TFO-mode allows greater value-range  $v$  and neighborhood size  $k$  (section 7.2), but the rules are limited to various types of totalistic rule (tcode and rcode), outer-totalistic rules, and reaction diffusion rules (not full rule-tables — rcode). 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, consists of all the states and merging links (trajectories) defined by the tree's root on the attractor (chapter 23).
- trajectory** ... the sequence of states 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) (section 26.2.2).
- update-tag** ... (from 0 to  $n-1$ ) in sequential updating (section 29.9 way that cell indexes are listed from left to right and updated).
- v2k3, v3k7 etc* ... the notation for showing the value-range and neighborhood size.
- value-range,  $v$**  ... the number of possible cell values, or colors, or internal states, or the size of the "alphabet" (chapter 7, sections 21.9,16.16).
- value-bias** ... the frequency of values randomly distributed in a rule-table (section 16.3.2), or in a seed or block (section 21.3.3).
- variability** ... of the entropy, from "Classifying rule space" (chapter 33), either min-max or sdev.
- wiring** ... the connections between network elements or cells, either local or non-local. For CA the *local* wiring creates the network geometry, 1d, 2d or 3d.
- wiring scheme** ... the actual wiring in the network (chapter 12).
- Z-parameter** ... A static parameters from the rcode-table giving the probability that the next unknown cell in the CA reverse algorithm is determined [31, 38, 48]. High  $Z$  predicts chaos, low  $Z$  predicts order (section 24.9, figure 24.10). To tune  $Z$  see sections 16.3 and 32.5.5.
-

# References

- [1] Adamatzky,A., (Ed) (2002) “Collision-Based Computing”, Springer, London.
- [2] A.Adamatzky,A., A.Wuensche, (2015) “On creativity of elementary cellular automata”, Complex Systems, Volume 22, Issue 4.
- [3] Adamatzky,A., J.Martnez (Eds.) (2016) “Designing Beauty: The Art of Cellular Automata”, Springer.
- [4] Albert,R., H.Jeong, and A-L.Barabasi, (2000) “Error and attack tolerance in complex networks”, Nature, vol 406.
- [5] Bilotta, E., A.Lafusa, and P.Pantano, (2003). “Is self-replication an embedded characteristic of the artificial/living matter?”, in Artificial Life VIII, eds. Standish and Bedau, MIT Press, 38-48.
- [6] Burraston, D. (2005) Structuring Cellular Automata Rule Space with Attractor Basins. Proceedings of Generative Arts Practice Symposium.  
<http://www.noyzelab.com/research/Burr-GAP05-final.pdf>
- [7] Burraston, D. (2006) Generative Music and Cellular Automata. PhD Thesis, Univ. Technology Sydney, Australia.
- [8] Burraston, D. (2011) Generative Music and Cellular Automata Bibliography, Leonardo Art, Science and Technology Bibliographies, 159-165, MIT Press.  
<http://leonardo.info/isast/spec.projects/generativemusic-biblio.pdf>
- [9] Byl,J., (1989) “Self-Reproduction in small cellular automata.” Physica D, Vol. 34, 295-299.
- [10] Conway,J.H., (1982) “What is Life?” in “Winning ways for your mathematical plays”, Berlekamp,E, J.H.Conway and R.Guy, Vol.2, chap.25, 817-850, Academic Press, New York.
- [11] Cortês,M., L.Smolin, (2014) “The universe as a process of unique events”, Phys. Rev. D 90, 084007.
- [12] 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, 344-353, Springer-Verlag.
- [13] Gomez-Soto, JM., and A.Wuensche, (2015) “The X-rule: universal computation in a non-isotropic Life-like Cellular Automaton”, JCA, Vol 10, No.3-4, 261-294.  
preprint: <http://arxiv.org/abs/1504.01434/>

- [14] Greenberg, J.M., and S.P. Hastings, (1978) "Spatial patterns for discrete models of diffusion in excitable media", *SIAM J. Appl. Math.* 34.
- [15] Gutowitz, H.A., ed., (1991) "Cellular Automata, Theory and Experiment", MIT press.
- [16] Harris, S.E., B.K. Sawhill, A. Wuensche, and S. Kauffman, (2002) "A Model of Transcriptional Regulatory Networks Based on Biases in the Observed Regulation Rules", *COMPLEXITY*, Vol.7/no.4, 23-40.
- [17] Hopfield, J.J. (1982) "Neural networks and physical systems with emergent collective computational abilities", *Proceedings of the National Academy of Sciences* 79:2554-2558.
- [18] Hordijk, W., J.P. Crutchfield, M. Mitchell, (1998) "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, 613-622.
- [19] Kauffman, S.A., (1969) "Metabolic Stability and Epigenesis in Randomly Constructed Genetic Nets", *Journal of Theoretical Biology*, 22, 437-467.
- [20] Kauffman, S.A., (1993) "The Origins of Order, Self-Organization and Selection in Evolution", Oxford University Press.
- [21] Langton, C.G., (1986) "Studying Artificial Life with Cellular Automata", *Physica D*, 22, 120-149.
- [22] Langton, C.G., (1990) "Computation at the Edge of Chaos: Phase Transitions and Emergent Computation", *Physica D*, 42, 12-37.
- [23] Li, W., (1989) "Complex Patterns Generated by Next Nearest Neighbors Cellular Automata", *Comput. & Graphics Vol.13, No.4*, 531-537.
- [24] Li, W., and N.H. Packard, (1990) "The Structure of the Elementary cellular automata rule space", *Complex Systems*, 4(3), 281-297.
- [25] Somogyi, R., and C. Sniegoski, (1996) "Modeling the Complexity of Genetic Networks: Understanding Multigenetic and Pleiotropic Regulation", *COMPLEXITY*, Vol.1/No.6, 45-63.
- [26] Shmulevich, I., H. Lhdesmki, E.R. Dougherty, J. Astola, W. Zhang, (2003) "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.
- [27] Walker, C.C., and W.R. Ashby, (1966) "On the Temporal Characteristics of Behavior in Certain Complex Systems", *Kybernetick* 3(2), 100-108.
- [28] Wolfram, S., (1983) "Statistical Mechanics of Cellular Automata", *Reviews of Modern Physics*, vol 55, 601-644.
- [29] Wolfram, S., ed. (1986) "Theory and Application of Cellular Automata", World Scientific.
- [30] Wolfram, S., (2002) "A New Kind of Science", Wolfram Media, Champaign, IL.
- [31] 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.



- [32] 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.
- [33] 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.
- [34] 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.
- [35] 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.
- [36] Wuensche,A., (1998) “Genomic Regulation Modeled as a Network with Basins of Attraction”, *Proceedings of the 1998 Pacific Symposium on Biocomputing*, World Scientific, Singapore.
- [37] Wuensche,A., (1998) “Discrete Dynamical Networks and their Attractor Basins”, *Proceedings of Complex Systems '98*, University of New South Wales, Sydney, Australia.
- [38] 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.
- [39] Wuensche,A., (2000) “Basins of Attraction in Cellular Automata; Order-Complexity-Chaos in Small Universes”, *COMPLEXITY*, Vol.5/no.6, 19-25. (based on an art exhibition in collaboration with Chris Langton)
- [40] 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.
- [41] Wuensche,A.,(2002), “Finding Gliders in Cellular Automata”, in “Collision-Based Computing”, ed. A.Adamatzky. Springer, London.
- [42] Wuensche,A., (2003) “Discrete Dynamics Lab: Tools for investigating cellular automata and discrete dynamical networks”. *Kybernetes* 32, 77-104.
- [43] Wuensche,A., (2004), “Self-reproduction by glider collisions: the beehive rule”, *Alife9 Proceedings*, eds J.Pollack et al., pages 286-291. MIT Press.
- [44] Wuensche,A., (2005) “Glider Dynamics in 3-Value Hexagonal Cellular Automata: The Beehive Rule”, *Int. Journ. of Unconventional Computing*, Vol.1, No.4, 2005, 375-398.
- [45] 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.
- [46] Wuensche,A., (2006), [http://uncomp.uwe.ac.uk/wuensche/multi\\_value/spiral\\_rule.html](http://uncomp.uwe.ac.uk/wuensche/multi_value/spiral_rule.html) Spiral-rule web page.
- [47] Wuensche,A., (2009), ”Cellular Automata Encryption: The Reverse Algorithm, Z-Parameter and Chain-Rules”, *Parallel Processing Letters (PPL)*, Vol 19, No 2, 283-297.

- [48] 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.
  - [49] Wuensche,A., (2010), "Complex and Chaotic Dynamics, Basins of Attraction, and Memory in Discrete Networks", ACTA PYSICA POLONICA B, Vol 3, No 2, 463-478.
  - [50] Wuensche,A., (2011), "Exploring Discrete Dynamics", Luniver Press.
-

# Index

- $k_{Lim}$ , 521
- $n_{Lim}$ , 8
- 1d neighborhood, 90, 91
- 1d space-time patterns
  - ring of cells, scrolling, 43
- 1d-2d-3d space-time patterns, 444–445
- 2d neighborhood, 91
- 2d+time space-time patterns, 446
- 3d neighborhood, 93–94
  
- acronym glossary, 6
- active cell, 173
- adjacency-matrix, 241–243
- Altenberg, Lee, 157
- ASCII encoding
  - rule, 168, 512
  - seed, 268, 269, 513
- asynchronous updating, 26, 393–395
- attractor basins, 6, 17
  - basin of attraction field, 35, 38
  - boundary conditions, 318–319
  - changes on-the-fly, 378–385
  - changing parameters, 36
  - computational limitations, 78
  - data, 331–346
  - display of, 318–330
  - drawing, 378–385
  - equivalent CA dynamics, 319–323
  - errors, 334, 380, 381
  - fan angle, 328
  - final options, 355–377
  - graphic conventions, 280–285
  - highlighting, 498
  - idea of, 280, 281
  - in-degree frequency, 292–296
  - layout, 309–317
  - layout in jump-graph, 236
  - mutation, 347–354
  - node display, 323–327
  - on-the-fly options, 308, 381
  - orientation, 328
  - output parameters, 286–308
  - pausing, 331–346
  - range of sizes, 38
  - revising the seed, 384
  - single, 39, 355–357
  - speed, 289, 308, 380, 383
  - subtree, 355
- attractor cycles, 283
- attractor histogram, 406–416
  - data, 411
  - density classification problem, 412
  - fuzzy attractors, 416
  - jump-graph, 415
  - pausing, 408
  - print/save, 412
  - rescaling, 411
  - sorting, 413
- attractor jump-graph, 221–243, 415
  - defining a block, 234
  - jump-table, 241–243
  - probabilistic “ant”, 234–236
- attractor states
  - print/save, 412
  
- backtracking in DDLab, 63
- backward space-time patterns, 37, 306–307
  - scrolling, 306
- bare attractor, 359
- basin of attraction field, 35, 38
  - in jump-graph PostScript, 238
  - layout in jump-graph, 236
  - or initial state, 11
  - or seed, 65, 66
- biased random wiring, 104–108
  - confine to zone, 105

- distinct, 107
- links to 3d layers, 107
- local treated as random, 105
- release wires from zone, 106
- same random everywhere, 108
- self-wiring, 106
- suppress periodic boundary conditions, 106
- bitmap graphics, 20, 517
- block, 173
  - density, 249–250
  - edges or solid, 188, 189, 193
- Boolean networks, 2
- boundary conditions
  - attractor basins, 318–319
  - null, 318–319
  - periodic, 318–319
  - space-time patterns, 393
- Burraston, Dave, 151
- Byl, J. — self reproducing loop, 32
- CA, *see* cellular automata
- canalyzing, 128, 135–141, 384, 464
  - $\lambda$  and  $P$  data, 305–306
  - Derrida plot, 136, 275
  - frequency, 138–141
  - homogeneous- $k$ , 137
  - mixed- $k$ , 138–140
  - single rcode, 207
  - single rule, 207
- cell color
  - by value or neighborhood, 388
- cell scale, 391
- cellular automata, 2, 6
- chain-rules, 128, 159
- classifying rule space, 435–437, 464, 475–496
  - sample file encoding, 496
  - sample plot data, 489
  - test data, 481
- color
  - basin edges, 329
  - nodes, 325
- command line arguments, 58
- compiling DDLab, 33
- complement rcode, 205
- complementary values, 167
- complex rules, 392
- compression of CA dynamics, 319–323
  - deactivate, 321
  - suppress trees, 323
- computational limitations
  - attractor basins, 78
- convert coordinates
  - 1d–3d, 95, 96
- Conway, John, 123, 157
- copying a rule to part of the network, 199
- copyright, 33
- Cywin, 57
- damage between two networks, 399
  - histogram, 402–406
- Das, Raja, 297, 408
- DDLab
  - accessing, 28–33
  - at SourceForge, 28
  - backtracking, 63
  - copyright, 33
  - latest versions, 28
  - license, 33
  - manual, 28
  - previous versions, 33
  - prompts, 63
  - quitting, 35
  - registration, 33
  - skipping forward, 35
  - source code, 28, 33
  - UNREGISTERED banner, 33, 59
  - web sites, 28
- DDN, *see* discrete dynamical networks
- deactivate equivalents, 321
- decimal rule limits, 153
- deleting a cell, 185
- density classification problem, 297, 407
  - attractor histogram, 412
- density of seed or block, 249–250
- density return map, 457–459
- density-bias, 122, 145
- Derrida plot, 177, 271–279
  - automatic sets of rules, 276
  - CA rule clusters, 277–279
  - canalyzing, 136, 275
  - coefficient, 276
  - completing parameters, 275
  - options, 272
  - parameters, 273

- directory, changing, 515
- discrete dynamical networks, 3, 4, 6, 53
  - measures, 9
  - parameters, 5, 9
- DOS, 4, 29, 35, 57
  - memory window, 63
  - mouse pointer, 62
  - resolution, 68
  - virtual memory, 63
- DOSBox, 29, 57
- dragging nodes or fragments
  - network-graph or jump-graph, 229–234
- duplicate the network and seed, 401
  
- edge color, 329
- effective  $k$ , 209
- effective  $k=0$ , 84, 125, 209
- Elementary Cellular Automata, 112
- emergent computation, 297
- encryption, 159
- entropy
  - of space-time patterns, 397, 456
  - single cell, 397
- entropy/density plot, 457
- equivalence classes, 205
- equivalent CA dynamics, 319–323
- equivalent rules with greater  $k$ , 208
- errors in attractor basins, 334, 380, 381
- exceeding normal limits, 12
- excitable media, 118
- exhaustive map, 203, 209
- exhaustive reverse algorithm, 26, 364–369
- exit DDLab, 64
- exLimits option, 67
  - $k_{Lim}$ , 72, 73
  - $k_{max}$ , 72
  - $n_{Lim}$ , 74
  
- fan angle in attractor basins, 328
- FIELD-mode, 11, 65, 66
  - network size limits, 74
- file encoding
  - $k$ -mix, 212
  - rule sample, 496
  - seed, 267
  - single rule, 161
  - wiring/rulemix, 211
- filing, 19, 199, 210–218, 511–516
  - $k$ -mix, 85, 89
  - attractor basin data, 339
  - data, 19
  - DDLab screen image, 59
  - directory, 515
  - graph layout, 239
  - list files, 516
  - loading  $k$ -mix networks, 217
  - loading constraints, 515
  - naming constraints, 511
  - network architecture, 210–218
  - network parameters, 19
  - networks and sub-networks, 213
  - prompt for saving and loading, 514
  - random map, 369
  - screen image, 20
  - seed, 257–262
  - wiring/rulemix, 211
- filtering space-time patterns, 208, 447, 451–453, 466
- first prompt, 65–70
- fixed borders, 466
- flashing cursor speed, 70
- font size/weight, 69
- forgetting, 497
- forwards-only
  - constraining to, 65, 66
  - TFO-mode, 11
  - totalistic, 8
- Fredkin, E., 158
- frozen generation size, 391
- frozen regions in space-time patterns, 447–450
  - frequency bins, 450
  - generations, 450
- full rule-table
  - binary, 112
  - multi-value, 114
- functions summary, 11–27
- fuzzy attractors, 24, 416
  
- $G$ -density, 298
  - against  $Z$  or  $\lambda_{ratio}$ , 301
  - against network size, 300
- game-of-Life, 47, 157, 269
  - outer-kcode, 118, 123–125
  - seeds, 31

- generation size
  - frozen, 391
  - input-entropy, 398
  - pattern density, 398
- geometry of network, 94–96, 101
- glider rules, 392, 434
  - creating, 434
- gliders
  - trail, 447
- glossary of acronyms, 6
- glossary of technical terms, 519–524
- Golly file format, 244, 246, 247, 269–270, 513
- graph layout
  - filing, 239
- graphics, 18
  - bitmap, 20
  - vector PostScript, 20, 517–518
- graphics setup, 35, 67, 468
  - DOSBox, 57
- Hamming distance, 271, 273, 465
- hexagonal neighborhood 2d, 90, 91
- highlighting, 497
  - states in attractor basins, 498
- histograms
  - 2d rule sample, 488
  - attractor, 406–416
  - canalyzing frequency, 138–141
  - cell wiring out-degree, 171, 178
  - damage, 402–406
  - in-degree, 292–296
  - partial pre-image stack, 358–364
  - skeleton, 416–421
  - wiring input, 200–201
  - wiring output, 200–201
- history limit, 356
- Hopfield, J., 497
- hybrid CA or HCA, 121
- hypercube wiring, 102
- in-degree frequency histogram, 292–296
  - cut-off, 292
  - data and prompts, 295
  - log plot, 296
  - rescaling, 296
- indexing
  - 1d networks, 95
  - 2d networks, 95
  - 3d networks, 96
- initial choices, 11
- initial state, *see* seed
- input-entropy, 397–456
  - classifying rule space, 475–496
  - generation size, 398
  - on-the-fly options, 399
  - single cell, 397
  - variability, 478
- input-frequency, 397
- inverse problem, 209
- invert kcode, 393
- invert rcode, 204
- isotropic rules, 128, 144, 425, 431, 432, 480
- jump-graph, 221–243, 415
  - activate, 224, 288
  - basins at nodes, 236
  - dragging nodes or fragments, 229–234
  - jump-table, 242
  - PostScript, 239
  - reminder, 225
- jump-table, 241–243
- $k$ , *see* neighborhood
- $k$ -matrix, 112–115, 132
- $k$ -mix, 12, 83–89, 100
  - at random, 87
  - changing the size, 198
  - copy rules automatically, 130
  - effective  $k$ , 209
  - effective  $k=0$ , 84, 125, 209
  - file encoding, 212
  - filing, 85, 89, 217–218
  - greater  $k$ , 208
  - increasing  $k_{max}$ , 88
  - $k_{Lim}$ , 84
  - max- $k$ , 208
  - normal distribution, 85
  - percentage of different  $k$ s, 86
  - power-law distribution, 86–87
  - reviewing, 88–89
    - within network architecture, 89
  - saving, 218
  - setting by hand, 85
  - showing the distribution, 87

- specifying, 84, 86
- transform, 203
- where all  $k$ 's are the same, 130
- with uniform  $k$ , 87
- $k=0$ , 84, 125, 209
- $k_{Lim}$ , 8, 11–13, 72–74, 116, 117, 177, 198
- $k_{max}$ , 88
- $k$ -totalistic rules, 116
- Kauffman, Stuart, 6, 16
- kcode, 114, 116
  - high  $v, k$ , 72
  - matrix, 117
  - mult-value, 115
  - outer-totalistic, 118
  - swapping values, 165
  - with increased  $k$ , 162
- kill a cell, 198
- Lafusa, Antonio, 110
- $\lambda$ -parameter, 122, 145, 298–305
  - weighted average, 196
- Langton, Chris, 32, 299
- layout of attractor basins, 309–317
  - pause, 315–317
  - preview, 309
- learning and forgetting, 27, 196, 497–510
  - aspiring pre-images, 502–504
    - parity, 503
  - highlighting only, 498
  - target state, 501
- Lesser, Mike, xiii
- Li, Wentian, 455
- Liapunov exponent, 22, 271, 276
- license, 3, 33
- Life-like rules, 123–125
- limit backward steps, 330, 358
  - to zero, 359
- limited subset
  - of rules, 121
- line spacing, 69
  - DOS, 69
- Linux within Windows, 57
- Linux-like operating systems, 4, 35, 56, 68
- Linux-like versions, 28
- list files, 516
- loading, *see* filing
  - constraints and warnings, 515
  - screen image in DDLab format, 59–60
  - sub-networks in set position, 216
- local CA neighborhood, 90–96
  - 1d, 91
  - 2d, 91
  - 2d hexagonal and square, 90
  - 3d, 93
- lookup-table, 112–117
- majority rules, 128, 131, 144, 154–156
  - shifted, 156
- matrix
  - kcode, 117
  - wiring, 171–173
- max- $k$ , 73, 88, 217
  - reducing to max network- $k$ , 208
- measures
  - on global dynamics, 23
  - on local dynamics, 22
- memory
  - insufficient, 74
- memory window DOS, 63
- min-max input-entropy variability, 478
- minimum  $k$ , 84
- Moore neighborhood, 13, 91, 157
- mouse pointer DOS, 62
- multi-value
  - 1d space-time patterns, 44
  - neighborhood matrix, 132
- mutation, 20–21, 347–354
  - of attractor basins, 347–354
  - on one screen, 313, 351
  - space-time patterns
    - current state, 439–440
    - rules, 430–433
    - wiring, 438
- $n_{Lim}$ , 74
- $n_{exhL}$ , 74
- negative equivalent rcode, 207
- neighborhood, 12, 83–96
  - $k$ -mix, 84, 100
  - 1d, 83, 91
  - 2d hexagonal or square, 91
  - 3d, 93
  - changing  $k$ , 198
  - effective  $k=0$ , 84, 125, 209

- greater  $k$ , 208
- $k$ -mix, 12, 83–89
- $k_{Lim}$ , 12, 84
- $k_{max}$ , 72
- pseudo, 5, 90
- size limits, 9
- neighborhood matrix, 112–115, 132
  - binary, 113
  - multi-value, 115
- network
  - 1d indexing, 95
  - 2d indexing, 95
  - 3d indexing, 96
- network architecture, 169–201, 384, 464, 469
  - filing, 210–218
  - network-graph, 223
  - sub-networks, 213
- network geometry, 94–96, 101
- network size, 12
  - 1d, 75–82, 104
  - 2d and 3d, 98, 99, 103, 104
  - limits, 8, 78, 356
- network-graph, 221–243
  - 1d, 2d, 3d, 94–96
  - adjacency-matrix, 241–243
  - defining a block, 234
  - dragging nodes or fragments, 229–234
  - options reminder, 223
  - PostScript, 239
  - probabilistic “ant”, 234–236
  - space-time patterns, 471–474
- neutral order
  - components, 27, 373
  - field, 376
  - subtree, 375
- node
  - colors, 325
  - display in attractor basins, 281, 323–327
- noisy updating, 45, 393–395
- nonlocal wiring, 90, 101–109, 438
  - reverse algorithm, 26
- null boundary conditions, 14, 426
  - attractor basins, 318–319
  - space-time patterns, 393, 438
- on-the-fly changes
  - attractor basins, 308, 378–385
    - options summary, 381–382, 422
  - bottom title bar, 424
  - classifying rule space, 475–496
  - space-time patterns, 422–474
    - current state, 439–440
    - options summary, 429
    - periodic-null wiring, 438
    - rules, 430–433
    - wiring, 438
- orientation of attractor basins, 328
- outer-kcode
  - game-of-Life, 118, 123–125
  - reaction-diffusion, 123
- outer-totalistic, 123–125
- output parameters
  - attractor basins, 286–308
  - space-time patterns, 386–421
- partial order updating, 26, 396
- partial pre-image histogram, 382
- partial pre-image stack, 358
  - local wiring, 362
  - nonlocal wiring, 362–364
- pattern density, 397, 456
  - generation size, 398
  - on-the-fly options, 399
  - single cell, 397
- pause options
  - layout of attractor basins, 315–317
  - space-time patterns, 462–474
- periodic boundary conditions, 90, 94
  - attractor basins, 318–319
  - space-time patterns, 393
  - suppress, 106
- platforms for DDLab, 3
- Post functions, 128, 133–134, 347
- PostScript files, 20, 517–518
  - amending linewidth, 518
  - cropping, 518
  - editing, 518
  - graph snapshot, 466
  - jump-graph, 239
  - jump-graph basins, 238
  - network-graph, 239
  - network-graph space-time pattern, 474
  - rule-table, 149, 151–152
  - seed, 254, 257–262, 465



- space-time patterns, 464, 469
- wiring graphic, 201
- wiring matrix, 172
- power-law distribution
  - of  $k$ , 86–87, 197
  - of outputs, 197
- pre-images
  - backwards space-time patterns, 306–307
  - early exit in pre-image fan, 379
  - exhaustive pairs, 365
  - in attractor basins, 280–281
  - in-degree frequency, 292–296
  - learning and forgetting, 196, 497–510
  - list of states, 344
  - partial pre-image stack, 361–362
  - reverse algorithms, 24–26
  - uniform states, 321
- printing
  - data, *see* terminal
  - screen image, 20, 59–62
    - in DOS, 62
    - Linux-like systems, 60–61
    - XV, 61
- probabilistic
  - “ant” in network or jump graph, 234–236
  - space-time patterns, 45, 394
  - updating, 393–395
- prompts in DDLab, 63
- pseudo-neighborhood, 5, 90, 187, 192
- quick start examples, 34–55
- random Boolean networks, 2, 6, 53
- random map, 3, 26, 367–369
  - data, 369
  - encoding, 369, 513
  - saving/loading, 369
- random number seed, 70
- random wiring, 90, 101–109
- range- $n$ , 1d, 75
- RBN, *see* random Boolean networks
- rcode
  - binary, 112
  - multi-value, 114
  - reaction-diffusion, 160
- reaction-diffusion, 112, 118–120
  - 2d, 118
  - 3d, 118
  - outer-kcode, 119, 123
  - rcode, 120, 160
  - threshold interval, 120
- recursive
  - inputs, 182
  - outputs, 182
- reflection equivalent of rcode, 207
- registration, 33
- return map
  - density, 457–459
  - value, 389, 457
- reverse algorithms, 24–26
  - 1d local wiring, 25
  - exhaustive, 364–369
  - nonlocal wiring, 26
    - reorder to optimize, 362
  - partial pre-image stack
    - local wiring, 362
    - nonlocal wiring, 362–364
- reverse engineering, 209
- ring of 1d cells, 43, 442, 472
- rotational symmetry, 319–323
- rule ASCII encoding, 168
- rule clusters, 277–279
- rule scheme, 5, 121–134
  - filing, 210–218
- rule value-bias, 146
- rule window, 165–166
- rule-table, 112–117
  - saved as seed, 149, 151, 512
- rulemix, 111, 121–134
  - alternate rules, 127
  - by hand, 128
  - canalyzing, 135
  - complete rules automatically, 130
  - copy rules automatically, 130
  - for large networks, or large  $k$ , 132
  - just one rule, 127
  - methods for setting, 125–127
  - random, 127
  - sequential blocks, 127
- rules, 14, 142–168
  - Altenberg, 45, 128, 144, 156, 157, 406
  - bias when classifying, 480
  - canalyzing, 128
  - chain, 128, 159

- clusters, 205
  - combinations, 111
  - decimal limits, 153
  - density-bias, 122, 145
  - encoding, 161
  - filing a single rule, 161
  - filing transformed rule, 204
  - game-of-Life, 123–125, 153, 157
  - in hex, 153
  - isotropic, 128
  - $\lambda$ -parameter, 122, 145
  - limited subset, 121
  - majority, 128, 131, 154–156
  - outer-totalistic, 111, 118
  - PostScript, 149, 151–152
  - reaction-diffusion, 112, 118–120
  - repeating last, 161
  - revising and copying, 199
  - saved as seed, 149, 151
  - set as bits or values, 148
  - show in terminal, 163
  - single, 121, 142–168
  - totalistic, 114
  - transform, 167, 199, 202–209
  - types, 110–120
- sample file encoding
- rule sample, 496
- saving, *see* filing
- attractor basin data, 339
  - attractor states, 412
  - density return map, 458, 463
  - rule sample plot data, 489
  - screen image, 59–62
    - Linux-like systems, 60–61
    - XV, 61
  - screen image in DDLab format, 59–60
- scale-free networks, 86, 197
- scatter plot of rule sample, 486
- screen image file of DDLab, 59–61
- screen-saver demo, 52, 298
- scrolling
- network-graph, 444, 446
  - ring of 1d cells, 43, 442
  - space-time patterns, 423, 459–460
- sculpting attractor basins, 27, 497–510
- seed, 15, 244–270
- 2d patch, 252
    - jump, 255
  - as bits or values, 251–262
  - ASCII, 268, 513
  - ASCII encoding, 269
  - complement, 167
  - density, 249–250
  - file encoding, 267
  - filing, 257–262, 267
  - Golly file format, 269–270, 513
  - in decimal, 263
  - in hex, 262
  - loading, 264
  - loading constraints, 264–267
    - or basin of attraction field, 11, 65, 66
  - PostScript, 254, 257–262, 465
  - random, 247–250
  - saved from rule-table, 149, 151, 512
  - value-bias, 249, 250
- SEED-mode, 11, 65, 66
- sequential updating, 26, 370–377
- attractor basin, 373
  - list all orders, 372–373
  - neutral order
    - components, 373
    - field, 376
    - subtree, 375
  - space-time patterns, 395, 430
- single basin
- history limit, 356
  - limit backward steps to zero, 359
  - of attraction, 356–357
- single rule, 142–168
- file,revise,transform, 384, 463–464
- size
- limits on neighborhood  $k$ , 9, 72–74
  - of font and line spacing, 69
  - of frozen generations, 391
  - of the DDLab screen, 18, 67–68
  - of the network, 12
    - 2d and 3d, 98, 99
  - limits, 8, 74, 78
    - limits for exhaustive algorithm, 74, 365
    - limits for FIELD-mode, 74
    - limits for seed in decimal, 263
  - of the rule-table, 112–117
  - of trailing time-steps, 398

- range of 1d, 75
- skeleton histogram, 416–421
  - data, 420
  - parameters, 416
  - pausing, 419
  - sorting, 421
- skip time-steps, 467
- solidify the rule, 205
- source code for DDLab, 3, 28, 33
- space-time patterns, 6, 16, 41–52
  - 1d PostScript, 469
  - 1d multi-value, 44
  - 1d scrolling, 391
  - 1d-2d-3d, 444–445
  - 2d+time, 446
  - analysis, 453
  - asynchronous updating, 393–395
  - attractor histogram, 406–416
  - backwards, 306–307
  - changes on-the-fly, 422–474
    - options summary, 422–429
  - classifying rule space, 435–437, 475–496
  - damage, 399
  - damage histogram, 402–406
  - density/entropy plot, 457
  - dimensions, 391
  - drawing, 422–474
  - duplicate the network and seed, 401
  - dynamic trace, 447
  - filtering, 208, 447, 451–453, 466
  - fixed borders, 466
  - frozen regions, 391, 447–450
    - frequency bins, 450
    - generations, 450
  - Hamming distance, 465
  - input-entropy, 397–456
    - generation size, 398
    - single cell, 397
  - network-graph layout, 466, 471–474
  - noisy updating, 45, 393–395
  - output parameters, 386–421
  - partial order updating, 396
  - pattern density, 397, 456
  - pause options, 462–474
  - PostScript, 464
  - probabilistic updating, 394
  - return map, 389
    - return map by value, 457
    - return map density, 457–459
  - scrolling, 459–460
  - sequential updating, 395, 430
  - skeleton histogram, 416–421
  - skip time-steps, 443, 467
  - sound, 443
  - speed, 382, 392, 424, 461, 468
  - state-space matrix, 389, 459
  - time-step pause, 392, 460
- speed
  - attractor basins, 289, 308, 380, 383
    - limitations, 78
  - space-time patterns, 382, 392, 424, 461, 468
- square neighborhood 2d, 90, 91
- standard deviation input-entropy variability, 478
- starting DDLab, 56–64
- state, *see* seed
- state-space matrix, 37, 289–292, 389, 459
- sub-networks, 16
  - loading in set position, 216
- subtree, 40, 285, 355
  - forward before backwards, 356
- subtree=basin, 337
- suppress equivalent trees and subtrees, 323
- swapping kcode values, 165
- t-totalistic rules, 117
- tcode, 114, 117
  - mult-value, 115
  - outer-totalistic, 118
- terminal, 56
  - exhaustive pairs, 366
  - attractor basin data, 339
  - attractor states, 412
  - jump-table or adjacency-matrix, 242
  - network data, 218–220
  - rule data, 163
- TFO-mode, 11, 12, 44, 65, 66
  - high  $v, k$ , 72
- time-steps
  - skip, 467
- title bar, 59, 66
- totalistic rules, 114, 115
  - constraining to run forwards-only, 65, 66
  - TFO-mode, 11

- transform rcode, 167, 199, 202–209
    - canalyzing, 207
    - complement, 167, 205
    - effective  $k$ , 209
    - exhaustive map, 209
    - greater  $k$ , 208
    - invert, 204
    - max- $k$ , 208
    - negative equivalent, 207
    - reflection equivalent, 207
    - solidify, 205
  - transient scale, 311–312
  - transient trees, 283
    - uniform states, 284
  - type of rule, 110
  - uniform states, 320
    - pre-images, 321
    - transient trees, 284
  - UNREGISTERED banner, 33
  - untangle wiring, 185
  - updating
    - asynchronous, 26
    - noisy, 45
    - partial order, 26, 396
    - probabilistic, 393–395
    - sequential, 26, 395
  - value return map, 390, 457
  - value-bias, 249
    - rule, 146
    - seed, 250
  - value-range, 5, 12, 71–74
    - limits on  $k$ , 72–74
    - limits on  $n$ , 74
    - prompt, 71
  - variability of input-entropy
    - min-max, 478
    - standard deviation, 478
  - vector graphics, 20, 517–518
  - VMware Player, 57
  - von Neuman neighborhood, 13, 91
  - Walker, Crayton, 277
  - wiring, 13, 101–109
    - 1d block, 182
    - 1d, 2d and 3d, 103
    - 2d block, 180–189
    - 3d block, 192–193
    - by hand, 108–109
    - graphic, 89, 173
      - 1d, 178–185
      - 2d, 186–189
      - 3d, 190–195
    - mouse pointer/click, 177
    - options reminder, 174–177
  - hypercube, 102
  - input frequency histogram, 200–201
  - local, 104
  - local treated as random, 104
  - nonlocal, 90, 101–109, 438
  - output frequency histogram, 200–201
  - periodic boundary conditions, 90, 94
  - periodic-null, 438
  - quick settings, 97–100
  - random, 90, 103–108
    - biased, *see* biased random wiring
  - reviewing, 100, 109
  - untangle, 185
  - wiring matrix, 171–173
- wiring scheme, 5, 101–109
    - filing, 210–218
    - loading, 98
  - wiring/rulemix: file encoding, 211–212
  - Wojtowicz, Mirek, 157
  - Wolfram, S., 112, 204
  - xterm, *see* terminal
  - XV, 61
  - Z-parameter, 145, 298–305, 433, 464
    - weighted average, 196