Discrete Dynamics Lab: Tools for investigating cellular automata and discrete dynamical networks*

Andrew Wuensche

Discrete Dynamics Inc.
7 Calle Andreita, Santa Fe, NM 87506, USA
wuensch@santafe.edu, www.ddlab.com

1 Introduction

Networks of sparsely inter-connected elements having discrete values and updating in discrete time (in parallel or sequentially) are central to a wide range of natural and artificial phenomena drawn from many areas of science; from physics to biology to cognition; to social and economic organization; to parallel computation and artificial life; to complex systems in general.

Such "decision making" networks are increasingly applied as idealized models in the study of complexity and emergence, and in the behavior of networks in general, including biomolecular networks such as neural and genetic networks[5, 7, 4, 11]. In addition, the networks themselves have intrinsic interest as mathematical, physical and computational systems with a large body of literature devoted to their study[8, 9, 1]. Because the dynamics is difficult to describe by classical mathematics, computer simulation is required, and there is a need for simulation software for non-experts in programming to model networks in their particular fields.

DDLab is an interactive graphics program aimed at addressing these issues, widely used in both research and education. The dynamics of a wide range of finite binary networks can be investigated, from Cellular Automata (CA) to Random Boolean Networks (RBN). The methods are currently being generalized for multi-state networks. There are currently versions of DDLab for Linux, Unix, Irix and DOS

As well as generating space-time patterns in one, two or three dimensions, DDLab is able to constructs attractor basins, graphs that link network states according to their transitions, analogous to Poincaré's "phase portrait" which provided powerful insights in continuous dynamics. A key insight is that the dynamics on the networks converge, thus fall into a number of "basins of attraction", which hierarchically categorize patterns of activation, "state-space", creating memory as a function of the network architecture[11, 13]. High convergence implies order, low convergence implies disorder or chaos[9]. The most interesting phenomena occur at the transition, sometimes called the "edge of chaos"[6].

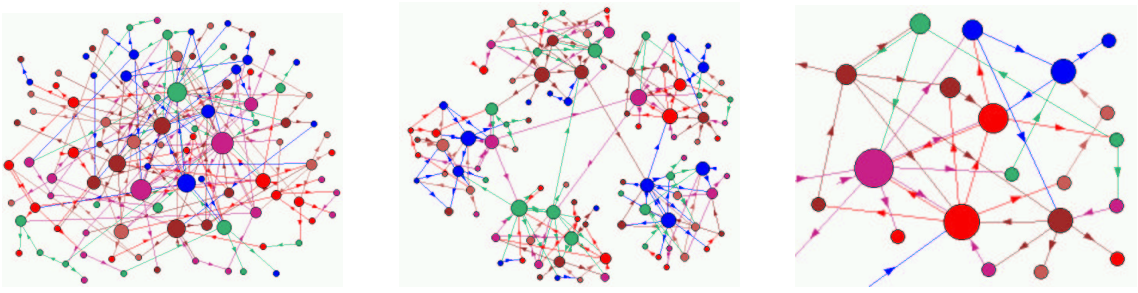


Figure 1: Hypothetical networks of interacting elements (size n=100) with an approximate power-law distribution of connections[2], both inputs (k) and outputs, which are represented by directed links (with arrows). Nodes are scaled according to k and average $k \simeq 2.2$. *left*: A fully connected network. *center*: A network made up of five weakly inter-linked n=20 sub-networks or modules. *right*: A detail of the top right sub-network.

*To appear in KYBERNETES

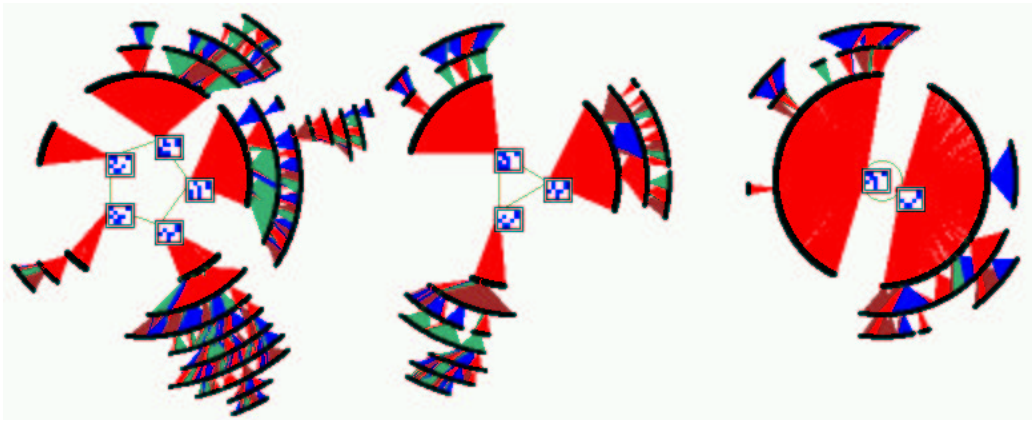
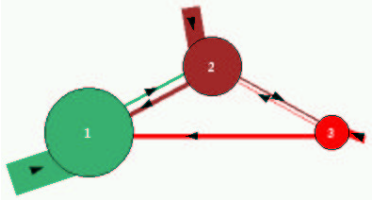


Figure 2: The basin of attraction field of the n=20 sub-network shown in detail in figure 1(c). The rules (input logic) were assigned at random. State-space (size $2^{20} \simeq$ 1.05 million) is partitioned into three basins of attraction. The attractor states are shown as 5x4 bit patterns. The table and diagram on the right show the probability of jumping between basins due to one-bit perturbations of their attractor states. P = attractor period, J = possible jumps ($P \times n$), V% is the basin "volume" as a percentage of state-space, and S% is the percentage of self-jumps for each basin. For example in basin 1, P=5, J=100 possible jumps, 6 of these jump to basin 2, none to basin 3, and 94 back to itself. All 3 basins are relativly stable because $S > V$. The diagram below the table, the "metagraph", shows the same data graphically. Node size reflects basin volume, link thickness percentage jumps, arrows the direction, and the short stubs self-jumps. The fraction of garden-of-Eden states in all three basins is 0.999+ indicating high convergence and order.

	1	2	3	P	J	V%	S%
1:	94	6	.	5	100	61.8	94.0
2:	12	44	4	3	60	28.6	73.3
3:	15	3	22	2	40	9.6	55.0



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". There are built in tools for designing and manipulating networks. A wide variety of measures, data, analysis and statistics are available. For small systems, its possible to compute and draw basins of attraction, and measure their convergence and stability to perturbation. For larger systems, basins of attraction can be investigated statistically. This article provides some general background, and gives the flavor of DDLab with a range of examples. The operating manual describes all of DDLab's many functions, and includes a "quick start" chapter. DDLab is available at `www.ddlab.com/` and mirror sites `www.santafe.edu/~wuensch/ddlab.html` and `www.cogs.susx.ac.uk/users/andywu/ddlab.html`.

DDLab remains free shareware for personal, non-commercial, users only. Any other users, including commercial users, companies, government agencies, research or educational institutions, must register and pay the license fee (see `www.ddlab.com/ddinc.html`).

2 Basins of Attraction

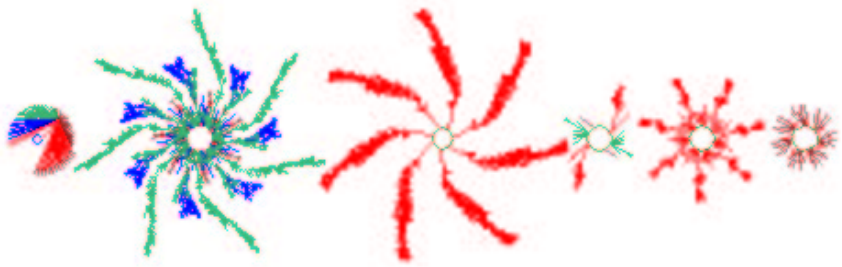
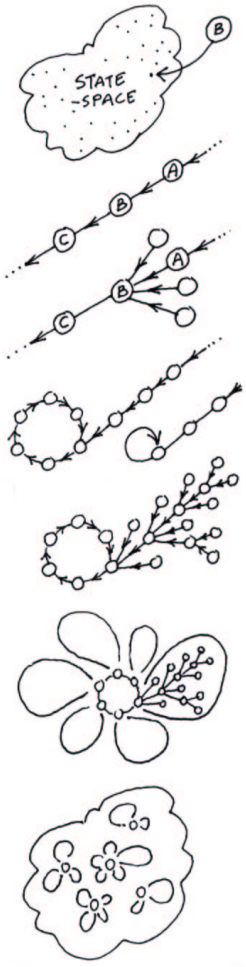


Figure 3: The basin of attraction field of a Cellular Automaton, $k = 3$, $n = 14$, rule 193, with equivalent basins suppressed.



For a network size n, an example of one of its states B might be $1010\ldots0110$. *State-space* is made up of all 2^n states, 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 on the network.

The state B may have other pre-images besides A, the total 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".

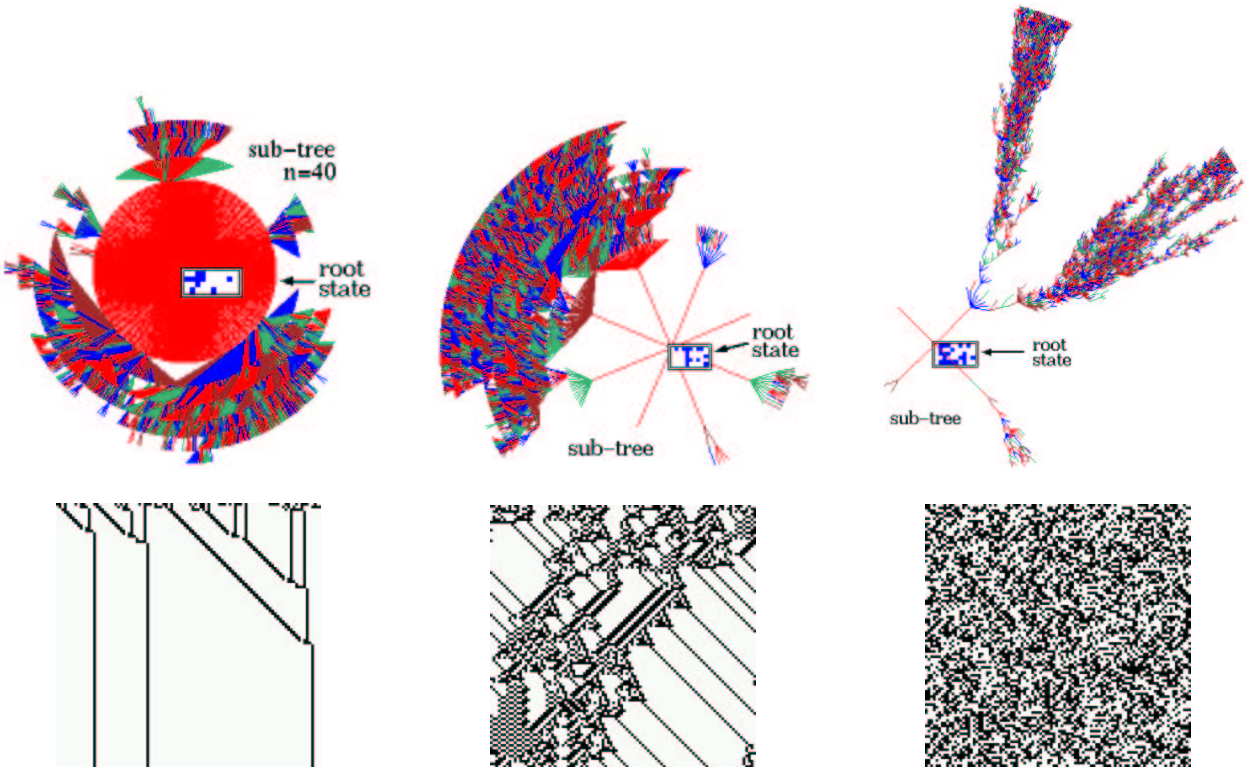
Now find every attractor cycle in state-space and construct its basin of attraction. This is the *basin of attraction field* containing all 2^n states in state-space, but now linked according to the dynamics on the network. Each discrete dynamical network imposes a particular basin of attraction field on state-space.

Figure 4: State-space and basins of attraction.

Figure 4 provides a summary of the idea of state-space and basins of attraction in discrete dynamical networks, sometimes called decision making networks. The dynamics depends on the connections and update logic of each element, which "decides" its next value based on the values of the few elements that provide its inputs, which might include self-input. The result is a complex web of feedback making the dynamics difficult to treat analytically, despite the simplicity of the underlying network. In fact, although the dynamics are deterministic, the future is in general unpredictable. Understanding these systems relies chiefly on computer simulation.

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 on/off state, and turn on or off according to the arriving signals. More abstractly, the network is made up of elements (or "cells" in CA), connected to each other by directed links or "wires", where a wire has an input and output terminal. A cell takes on a value of either 0 or 1, and transmits this value down its output wires. Its own value is updated as a function of the values on its input wires, the cell's rule. Updating is usually done in parallel, in discrete "time-steps", but may also be sequential in a predetermined order. RBN do not necessarily restrict rules and wiring, whereas CA cells are wired into a strictly regular lattice and obey one universal rule.



Ordered rule 01dc3610 | Complex rule 6c1e53a8 | Chaotic rule 994a6a65

Figure 5: Ordered, complex and chaotic dynamics of 1d CA are illustrated by the space-time patterns and subtrees of three typical k=5 rules (shown in hex). The bottom row shows the space-time patterns from the same random initial state. The bit-strings (n=100) of successive time-steps (represented by white and black dots) are shown horizontally one below the other, i.e. time proceeds down. Above each space-time pattern is a typical sub-tree for the same rule. In this case n=40 for the ordered rule, and n=50 for the complex and chaotic rules. The root states were reached by first iterating the system forward by a few steps from a random initial state, then tracing the subtree backwards. Note that the convergence in the sub-trees, their branchiness or typical in-degree, relates to order-chaos in space-time patterns, where order has high, chaos low, convergence.

A multi-value version of DDLab is currently being developed, where the "value-range" instead of being just 2 (i.e. 0,1), can be selected anywhere from 2 to 8.

The DDlab user is able to create these networks, and graphically represent and analyze both the networks themselves and the dynamics resulting from the changing patterns of elements/cells (the term is used interchangeably). The networks, whether CA or RBN, may be set up in one, two or three dimensions. Network updating may be sequential as well as parallel, noisy as well as deterministic. This is the system in a nutshell. It remains to set up the network according to the required parameters,

- The value-range v, from 2 to 8, which will be a new option in the forthcoming multi-value version of DDLab. In the current binary version v=2
- The number of elements, the system size, n.
- How the elements are arranged in space: a 1d, 2d or 3d array with axial dimensions i, j, h, or some other arrangement. This network "geometry" may have real meaning (depending on the "wiring scheme" below), or it may simply allow convenient indexing and representation.
- The number of input wires, k, to each element, or the "k-mix" if k is not homogeneous. k may vary from 0 to 13.

- The "wiring scheme". Defining the location of the output terminals of an element's input wires, the "neighborhood". Cellular automata have local "nearest neighbor" wiring, an identical neighborhood template throughout the network. Random Boolean networks may have a completely arbitrary wiring scheme, where each element has a "pseudo-neighborhood" assigned at random, or the wiring scheme may be biased in some way, for example, by confining an element's pseudo-neighborhood close to itself. The wiring scheme also defines boundary conditions. CA wiring usually has "periodic boundary conditions", where an array's edges wrap around to their opposite edges. Network wiring can also be set up to correspond to a regular n-dimensional hypercube, where $k = log_2(n)$.

- The "rule scheme", the rules, or Boolean (or multi-state) functions in the network. Each element applies a rule to its inputs to compute its output. Usually this is made into a look-up table, the "rule-table", listing the outputs of all possible input patterns. Cellular automata have the same rule throughout the network. Random Boolean networks may have a completely arbitrary rule scheme, or again, it may be restricted or biased in various ways.

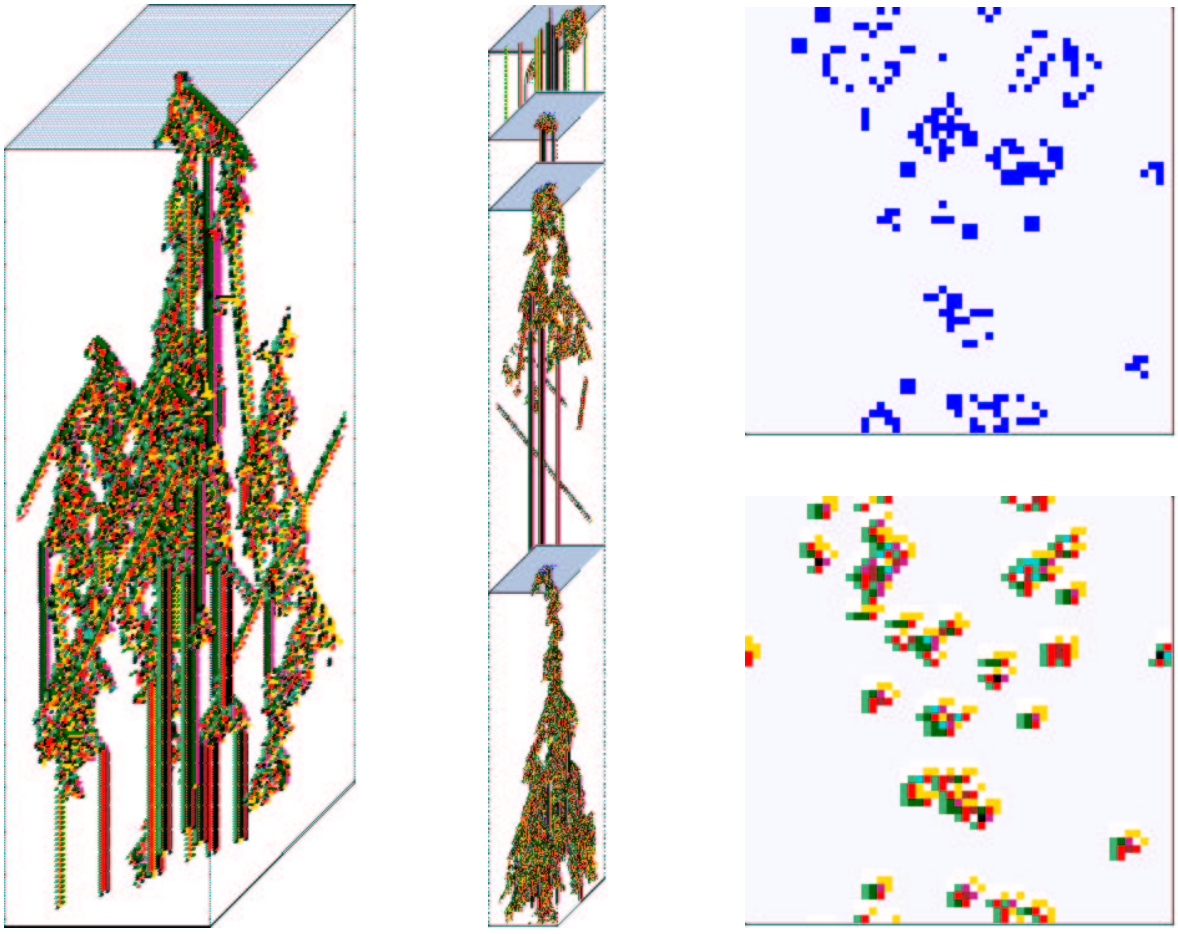


Figure 6: Space-time pattern of the 2d game-of-Life[3], (k=9, $n = 55 \times 55$) in a 3d isometric projection. 2d time-steps stack below each other, and are shown as if looking up at a transparent shaft. *left*: Starting from the "r-pentomino" seed. *center*: Re-scaled to the smallest scale, new seeds set at intervals. *upper right*: A particular state (time-step). *lower right*: A particular state colored according to the neighborhood look-up instead of the value.

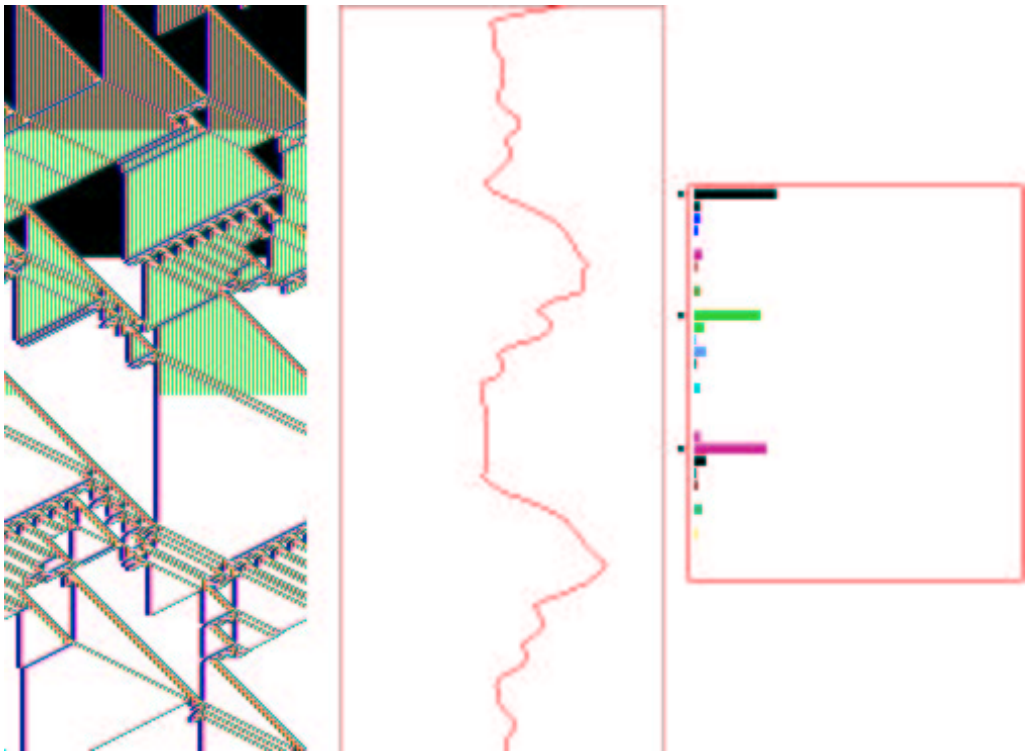


Figure 7: A space-time pattern of a complex 1d CA, $k = 5$, hex rule e9 f6 a8 15, $n = 150$. About 360 time-steps, including some analysis shown by default. *left*: The space-time pattern colored according to neighborhood look-up, and progressively "filtered" on-the-fly at three times, suppressing the 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.

4 Space-time patterns and attractor basins

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

Running forwards generates the network's space-time patterns from a given initial state. Many alternative graphical representations of space-time patterns, and methods for gathering and analyzing data, are available to illustrate different aspects of local network dynamics, including "filtering" to show up emergent structures more clearly as in figure 3.

Running "backwards" generates multiple predecessors rather than a trajectory of unique successors. This procedure reconstructs the branching sub-tree of ancestor patterns rooted on a particular state. States without predecessors are disclosed, the so called "garden-of-Eden" states, the "leaves" of the sub-trees. Sub-trees, basins of attraction (with a topology of trees rooted on attractor cycles), or the entire basin of attraction field (referred to collectively as "attractor basins") can be displayed as directed graphs in real time, with many 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.

It can be argued that attractor basins represent the network's "memory" by their hierarchical categorization of state-space. Each basin is categorized by its attractor and each sub-tree by its root. Learning/forgetting algorithms allow attaching/detaching sets of states as predecessors of a given state by automatically mutating rules or changing connections.

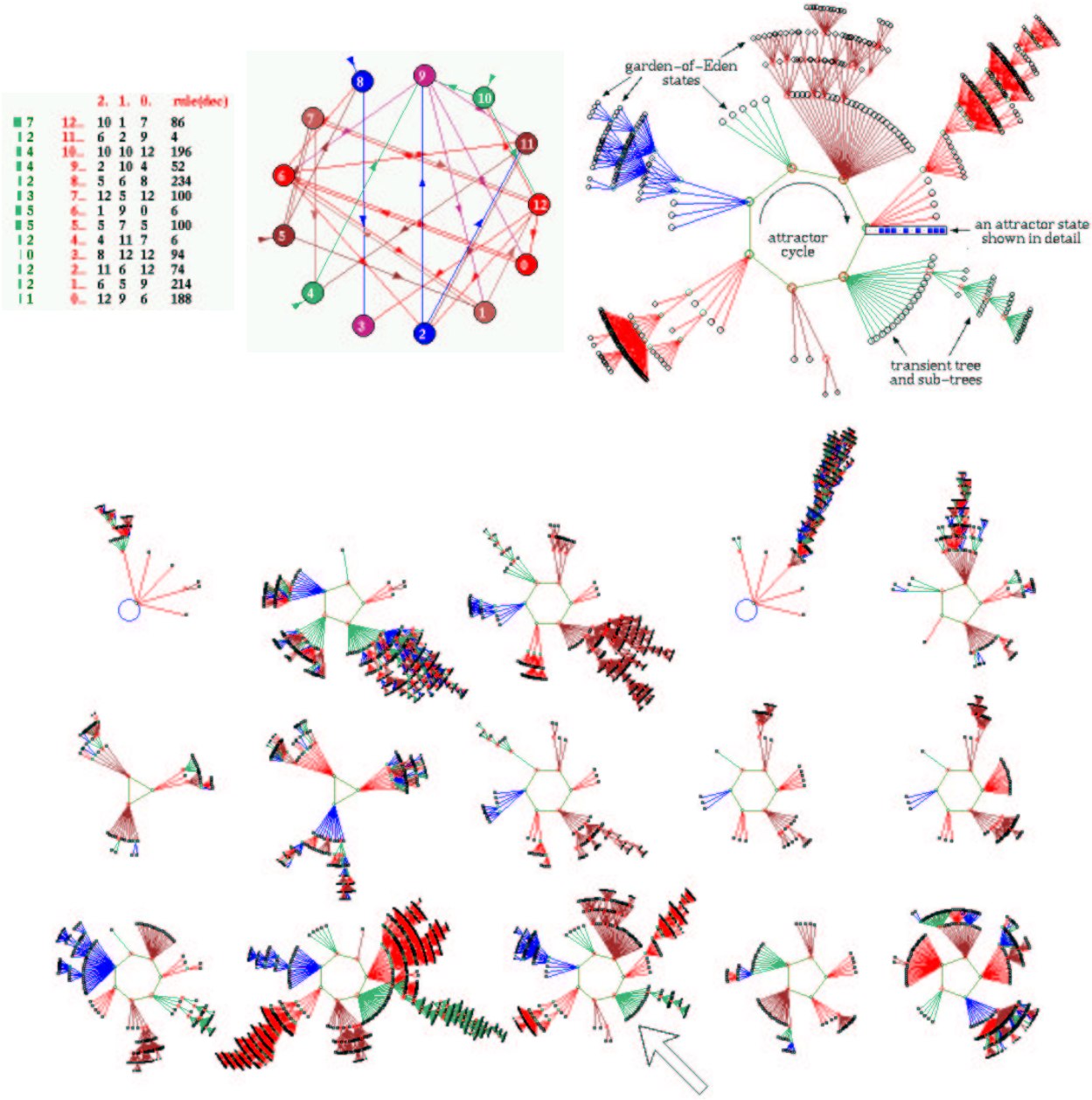


Figure 8: A small random Boolean network, n=13, with homogeneous k=3 wiring, though some elements have more than one input from the same element. *top left*: The network matrix, which fully defines wiring/rule scheme. *top center*: The network shown as a directed graph with numbered nodes, self-links are short arrows sticking into nodes. *top right*: One of the basins of attraction. The basin links 604 states, of which 523 are garden-of-Eden states. The attractor period is 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. *bottom*: The basin of attraction field. The $2^{13} = 8192$ states in state-space are organized into 15 basins, with attractor periods ranging between 1 and 7, and basin volume between 68 and 2724. The arrow points to the basin shown in more detail.

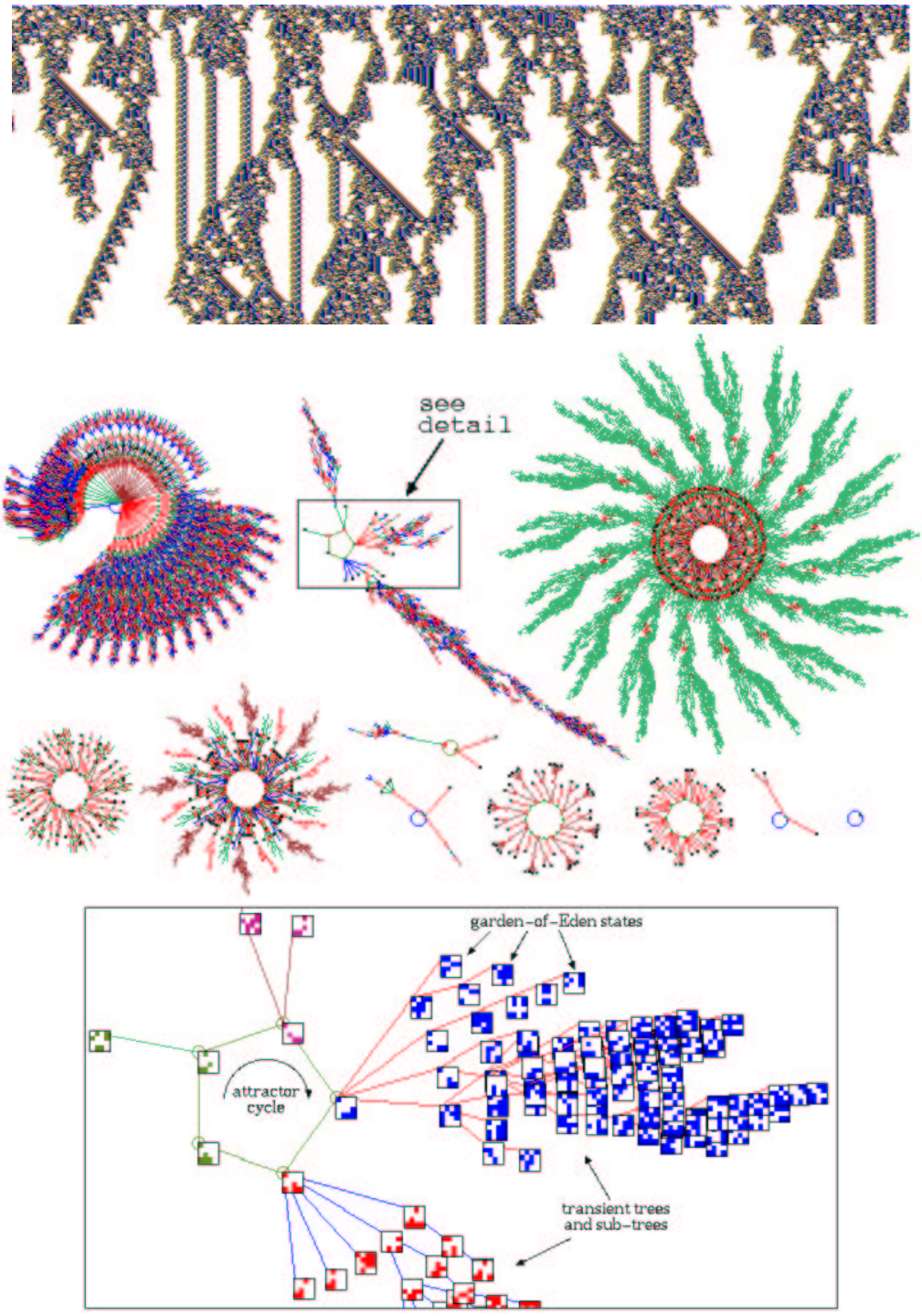


Figure 9: *top*: The space-time pattern of a 1d complex CA with interacting gliders[13], n=700, k=7, showing 308 times-steps from a random initial state. *center*: The basin of attraction field for the same rule, n=16. The 2^{16} states in state-space are connected into 89 basins of attraction, but only the 11 non-equivalent basins are shown, with symmetries characteristic of CA. *bottom*: A detail of the second basin in the basin of attraction field, where states are shown as 4×4 bit patterns.

5 DDLab user interface and platforms

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

Changes can be made on-the-fly, including changes to rules, connections, current state, scale, and alternative presentations highlighting different properties of space-time patterns and attractor basins. Networks of whatever dimension can be interchangeably represented in 1d, 2d, and 3d, as well as in a 2d isometric projection.

The network architecture, states, data, and the screen image can be saved and loaded in a variety of tailor-made file formats.

The user interface allows setting, viewing and amending network parameters, and running space-time patterns or attractor basins, 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 in the sequence, and in some cases skip forward. A flashing cursor indicates the current prompt. 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.

Compiled versions of DDLab are currently available for Linux, Unix, Irix and DOS. The source code is written in C. It may be made available on request, subject to various conditions.

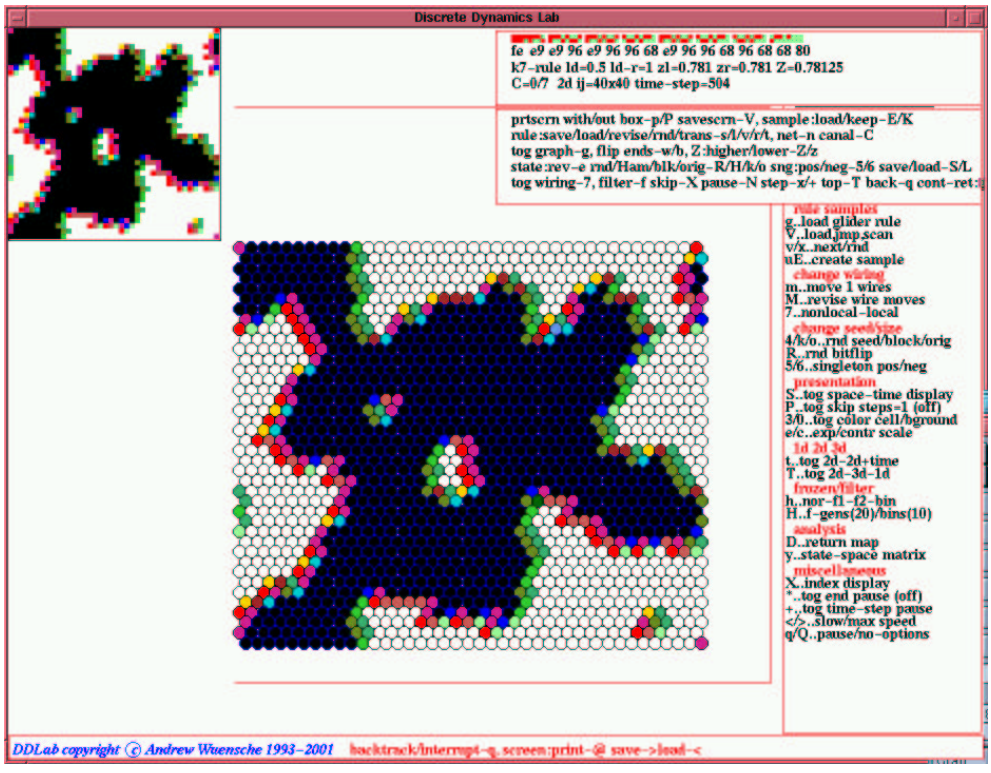


Figure 10: 2d CA space-time patterns shown according to the current network graph layout, in this case the default triangular grid for k=6 or 7. The network is 40x40, k=7, with a modified majority rule, totalistic code 232. The normal 2d space-time patterns are at the top left of the screen. The top right and right windows give rule details, interrupt options, and on-the-fly options.

6 The neighborhood k or k-mix

The size of the "neighborhood", the number of inputs each cell receives, k, can vary from 0 to 13. 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. A k-mix may be saved/loaded from a file, but is also implicit in the wiring scheme.

1d. k=0-13. The extra asymmetric cell in even k is on the right. The wiring is shown between two time-steps.

2d, k=2-13 (k=0-1 as in 1d). Note that k=6 and k=7 define an effectively triangular grid by changing between odd and even rows. The classical von Neumann and Moore neighborhoods are indicated.

3d, k=6-13 (k=0-5 as in 2d), shown as if looking up into an axonometric cage.

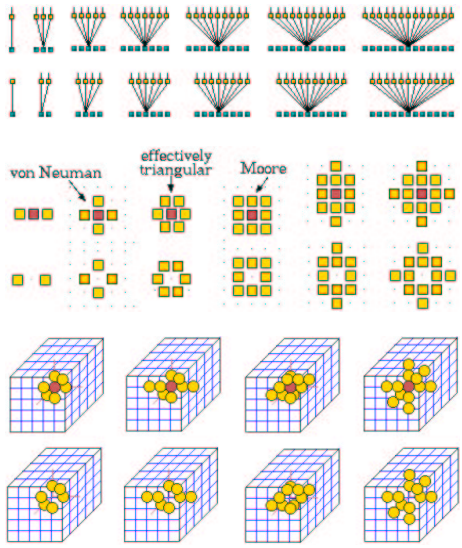


Figure 11: 1d, 2d and 3d CA neighborhood templates defined in DDLab. In 2d and 3d, to maximize symmetry, even k does not include the central target cell.

7 Wiring

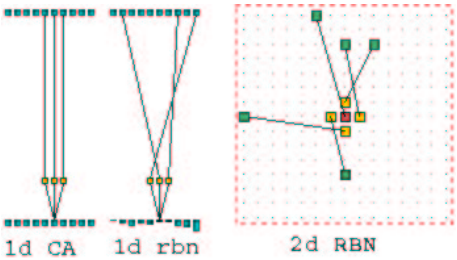
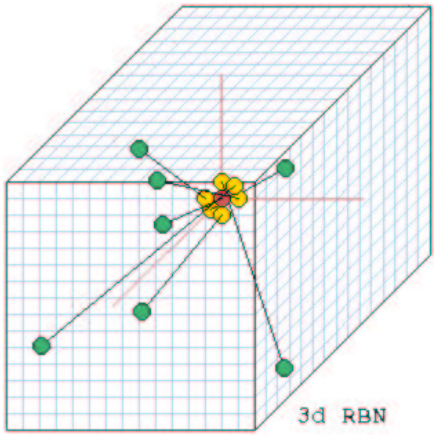


Figure 12: Network wiring. *left:* 1d, k=3, CA and RBN. The wiring is shown between two time-steps. *center:* 2d, k=5. *right:* 3d, k=7. In RBN, cells anywhere in the network are wired back to each position in the "pseudo-neighborhood".



The network's wiring scheme (i.e. its connections) has default settings for regular CA (for 1d, 2d and 3d) for each neighborhood size, k=0 to 13, as shown in figure 11. Wiring can also be set at random (non-local wiring), with a wide variety of constraints and biases, or by hand. A wiring scheme can be set and amended just for a predefined sub-network within the network, and may be saved/loaded from a file.

In most cases regular 2d wiring defines a square grid on the torus, and includes the von Neumann and Moore neighborhoods of 5 and 9 cells. However the 6 and 7 cell regular 2d neighborhood is wired according to a triangular grid. Regular 3d wiring defines a cubic grid with periodic boundary conditions.

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

of wiring scheme, for example rows in 2d and layers in 3d. The wiring can be biased to connect designated rows or layers.

The network parameters can be displayed and amended in a 1d, 2d or 3d graphic format as in figure 12, in a "spread sheet" as in 20, or as a network graph which can be rearranged in various ways, including dragging nodes with the mouse as in figures 1 and 22.

8 Rules

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

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

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

9 The initial network state, seed

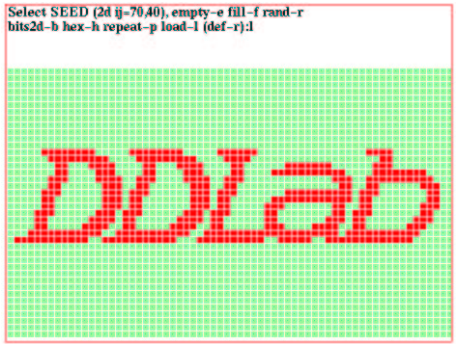


Figure 13: Drawing a 2d initial state with the mouse or keyboard. Start with a clean slate, all 0s or 1s, or some other pattern. Draw 1s, Os, vertical or horizontal lines, with the mouse or keyboard. Move without drawing with the arrow keys, or by clicking a new position with the mouse. Sub-patterns saved earlier can be loaded into specified positions within the main pattern.

An initial network state (the seed) is required to run the network forward and generate space-time patterns. A seed is also required to run backwards to generate a sub-tree or single basin. A basin of attraction field does not require a seed.

As in setting a rule, there are a wide variety of methods for defining the seed, in decimal or hex, as a bit pattern in 1d, 2d or 3d, or at random with various constraints or biases such as density. The seed can be saved/loaded from a file. The bit pattern method is a mini drawing program, using the mouse (or keyboard) to draw cell values (0,1), particularly useful for 2d and 3d.

10 Networks of sub-networks

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

The parameters of the sub-networks can be totally different from the base network, provided the base network is set up appropriately, with the right kind of memory to accommodate the sub-network. For

example, to load an RBN into a CA, the CA may need be set up as if it were an RBN. To load a mixed-k sub-network into single-k base network, k in the base network needs to be at least as big as the biggest k in the sub-network. Options are available to easily set up networks in this way. Once loaded, the wiring can be fine-tuned to interconnect the sub-networks as in figure 1.

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.

11 Presentation options for space-time patterns

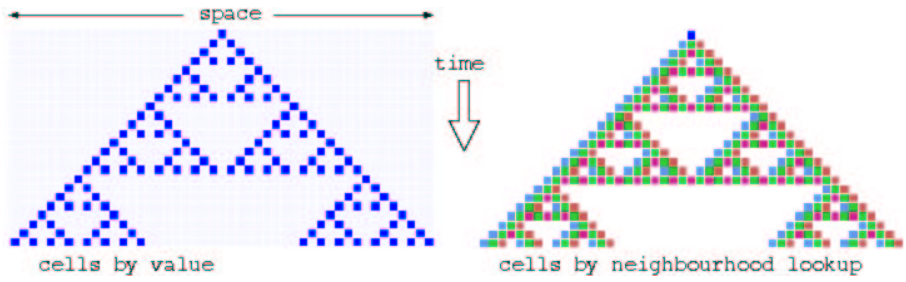
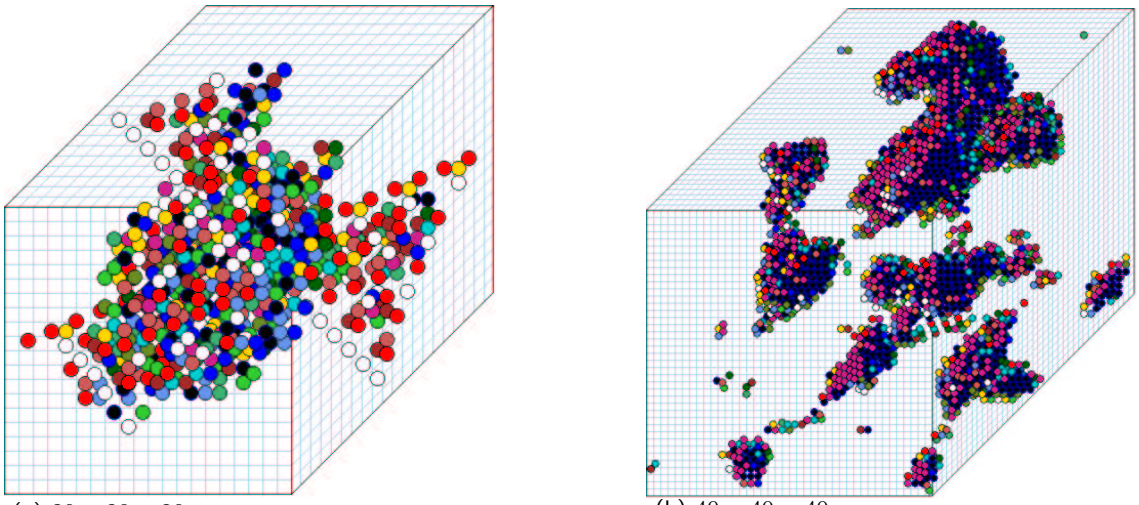


Figure 14: Space-time patterns of a 1d CA (n=24, $k = 3$, rule 90). 24 time-steps from an initial state with a single central 1. Two alternative presentations are shown. $Left$, cells by value, light=0 dark=1. $Right$, cells colored according to their look-up neighborhood.



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

(b) $40 \times 40 \times 40$,
k7 totalistic code=11101000

Figure 15: Examples of 3d CA with a k=7 neighborhood arranged as a 3d cross. The projection is axonometric seen from below, as if looking up at the inside of a cage. Cells are shown colored according to neighborhood lookup by default for a clearer picture (instead of by value: 0,1). (a) shows the evolution of a 3d CA, $n = 20 \times 20 \times 20$, with a randomly selected rule. The initial state is a "singleton seed", a single on cell in an otherwise empty array. (b) shows a 3d CA, $n = 40 \times 40 \times 40$ (the maximum size DDLab supports), The initial state was set at random, but with a bias of 45% of on cells.

Many options are provided for the presentation of space-time patterns. Again, many of these settings can be changed "on the fly". Cells in space-time patterns are colored according to their value (0,1) or alternatively according to their neighborhood at the previous time step, the entry in the look-up table that determined the cell's value. A key press will toggle between the two. Space-time patterns can be progressively filtered to suppress cells that updated according to the most frequently occurring neighborhoods (see figure 3), exposing "gliders" and other structures. 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. The emergence of such frozen elements is associated with "canalizing inputs", and underlies Kauffman's RBN model of gene regulatory networks[5, 4].

2d networks such as the "game-of-Life" can be displayed simply on a 2d grid, and also as a space-time pattern (2d+time) in a 3d isometric projection as in figure 6. 3d networks are presented 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.

Concurrently with these standard presentations, space-time patterns can be displayed in a separate window according to the network graph layout, including various default layouts as in figure 10. For example, a 1d space-time pattern, in fact a ring of cells because of periodic boundary conditions, can be shown in a circular layout.

12 Presentation options for attractor basins

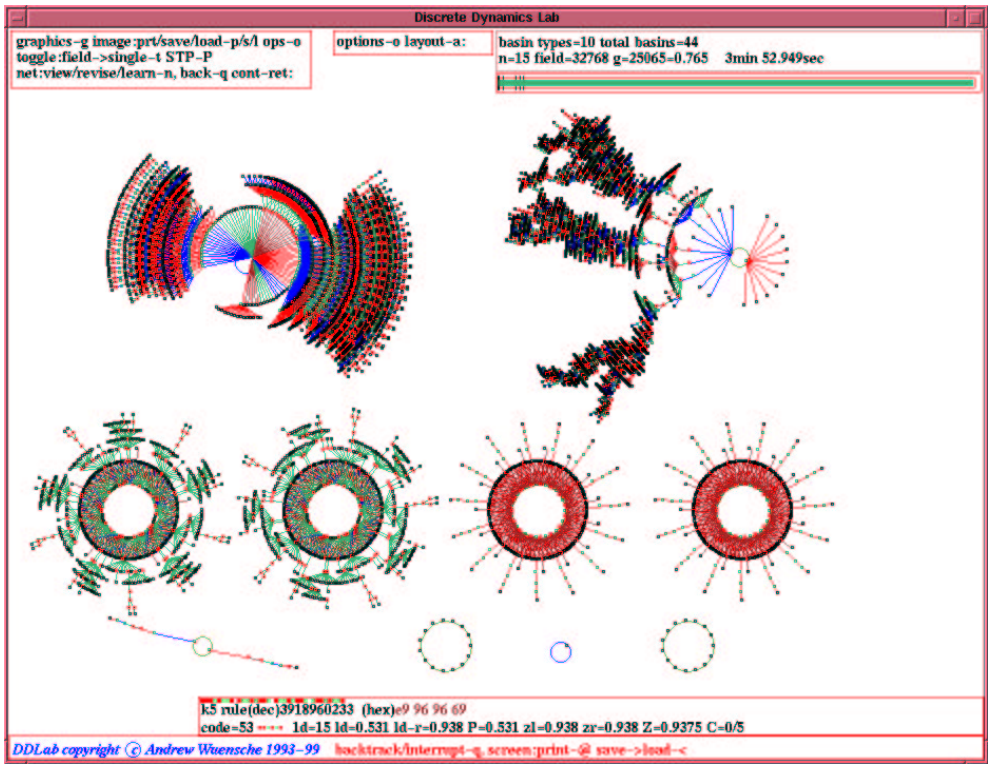


Figure 16: The DDLab screen showing a basin of attraction field. This example is for a 1d CA, n=15, k=5 totalistic code 53. To achieve this layout, a pause was selected after each basin, and the position and spacing of basins were amended on the fly. Various typical indications on the screen are as follows, *top left:* An options window. *top center:* The window for amending the layout on-the-fly. *top right:* The data window. *below top right:* The progress bar. *bottom center:* The rule window. *foot of screen:* The title bar.

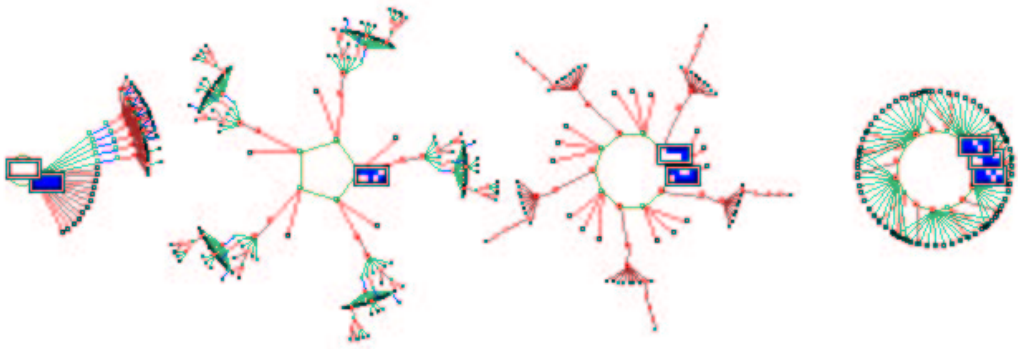


Figure 17: Highlighting all non-equivalent attractor states as a 2d bit pattern for the basin of attraction field of a 1d CA, k=3, n=10, rule (dec) 111. In this example other nodes are shown as spots.

Options for attractor basins allow the selection of the basin of attraction field, a single basin (from a selected seed), or a sub-tree (also from a seed, as in figure 5). Because a random seed is likely to be a garden-of-Eden state, to generate sub-trees an option is offered to run the network forward a given number of steps to a new seed before running backward. This guarantees a sub-tree with at least that number of levels.

Options (and defaults) are provided for the layout of attractor basins, their size, position, spacing, and type of node display (as a spot, in decimal, hex or a 1d or 2d bit pattern, or none). The basin of attraction field can also be displayed according to the meta-graph layout as in figure 18, with or without the meta-graph itself.

Regular 1d and 2d CA produce attractor basins where sub-trees and basins are equivalent by rotational symmetry. This allows "compression" of basins (by default) into non-equivalent prototypes, though compression can be turned off. Attractor basins are generated for a given system size, or for a range of sizes. As attractor basins are generating, the reverse space-time pattern can be 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.

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 "screensave" demo option shows new basins continually growing at random positions.

13 Filing

DDLab allows filing a wide range of file types, including network parameters, data, and the screen image. 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. Data on attractor basins, at various levels of detail can be automatically saved. A file of "exhaustive pairs", made up of each state and its successor, can be created

Various data including mean entropy and entropy variance of space-time patterns can be automatically generated and saved. This allows a sorted sample of CA rules to be created, discriminating between order, complexity and chaos[13]. A large collection of complex rules, those featuring "gliders" or other large scale emergent structures, can be assembled. Pre-assembled files of 1d CA rules sorted by this method are provided with DDLab, for k=5, 6 and 7.

The screen image is saved and loaded using an efficient home-made compressed format which is only applicable within DDLab. Alternatively, the DDLab window or part of it can be saved and printed using an external screen grabber.

Figure 18: The basin of attraction field (in figure 8) redrawn within the nodes of the meta-graph. The meta-graph shows the probability of jumping between basins due to single bit-flips to attractor states. Nodes representing basins (shown inside each node) are scaled according the number of states in the basin (basin volume). Links are scaled according to both basin volume and the jump probability. Arrows indicate the direction of jumps. Short stubs are self-jumps. Note that the meta-graph itself can be suppressed, making this an alternative flexible method for positioning basins.

14 Mutations

A wide variety of network "mutations", as well as changes in presentation can be be made.

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

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 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. The complete set of one-bit mutants of a rule can be displayed on a single screen as in figure 19.

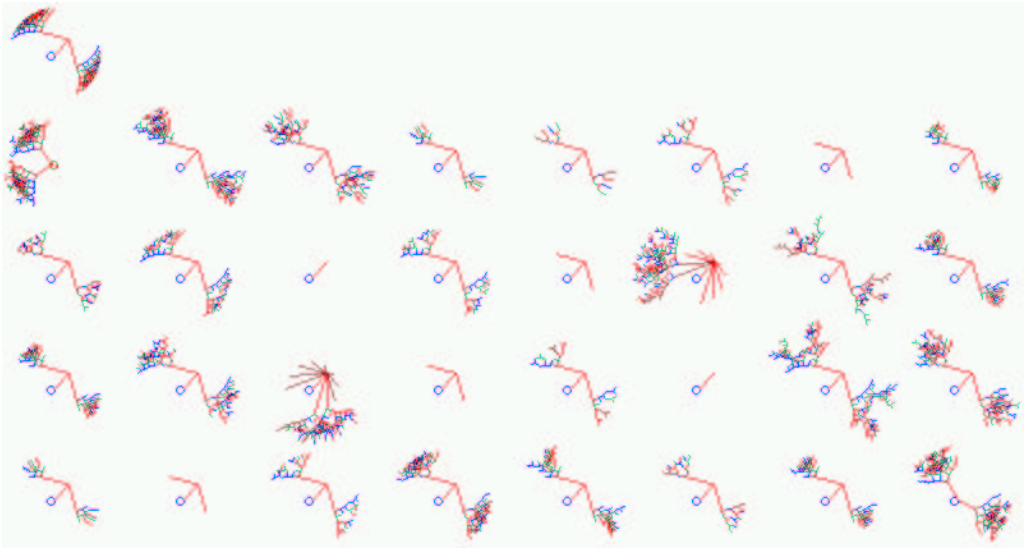


Figure 19: 32 mutant basins of attraction of the k=3 rule 195 (n=8, seed all 0s). *top left*: The original rule, where all states fall into just one very regular basin. The rule was first transformed to its equivalent k=5 rule (f00ff00f in hex), with 32 bits in its rule-table. All 32 one-bit mutant basins are shown. If the rule is the genotype, the basin of attraction can be seen as the phenotype.

15 Reviewing network architecture

DDLab provides two methods for reviewing network architecture, both wiring and rules. From the wiring matrix (figure 20) and network architecture graphic (figure 21), which can be displayed in 1d, 2d or 3d, the network's connections and rules can be examined, changed, and tailored to requirements, including biased random settings to pre-defined parts of the network. These are very flexible methods, and for RBN its usually easier to set up a suitable dummy network initially, then tailor it here. Network connectivity measures from the network architecture graphic include the following,

- Average k (inputs), and the number of reciprocal links, and self links.
- Histograms of the frequency distribution of inputs (i.e. k), outputs, or both (i.e all connections) in the network.
- The recursive inputs/outputs to/from a network element, whether direct or indirect, showing the "degrees of separation" between elements.

The second method is an adjacency matrix and network graph (see figures 1 and 22) that looks just at the network connections, nodes linked by directed edges. It does not allow changes to the underlying network, but includes very flexible methods for changing the presentation, rearranging and unraveling the graph.

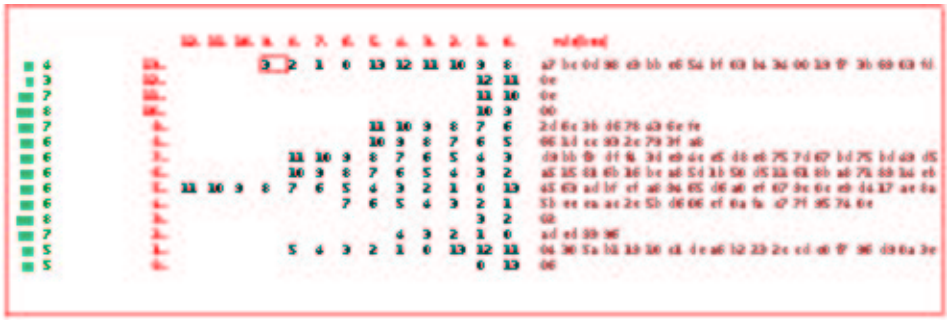


Figure 20: The wiring matrix for a mixed k network with random wiring. n=14, k=2-13, with rules set. k-12...0, indexes columns, n-13...0, indexes rows. The column on the left shows the "out-degree" of each cell, the number of output wires that link to it, also shown as a histogram. If rules have been set, they are shown in hex (as much as will fit) on the right, in the column "rule(hex)". Its possible to move around the wiring matrix as in a spread-sheet to change wiring settings.

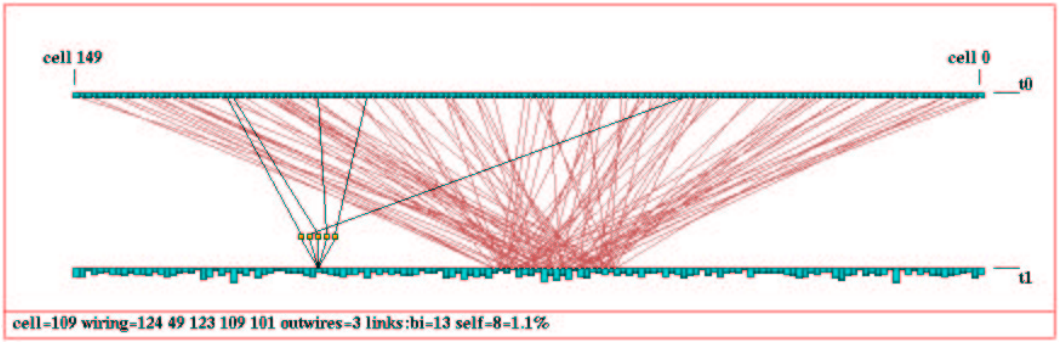
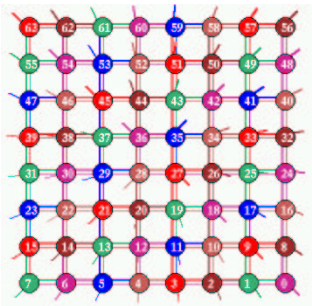


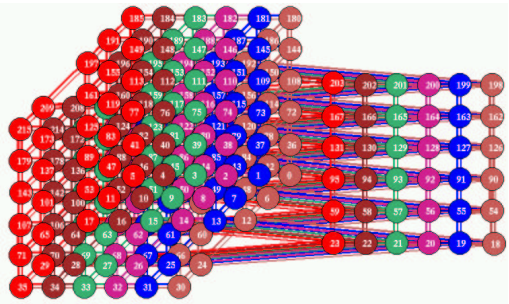
Figure 21: The 1d wiring graphic, showing wiring to a block within a 1d network. k=5, $n = 150$. The block was defined from cell 60-80. Revisions to rules and wiring can the be confined just to the block. The 1d wiring graphic can also be shown as a circle. The "active cell" (109) is still visible, and can be moved as usual. Its rule is shown in a lower window.

For example, single nodes, connected fragments, or whole components, can be dragged with the mouse to new positions with "elastic band" edges. Fragments depend on inputs, outputs, or both, and the distance of fragment links from a node can be defined.

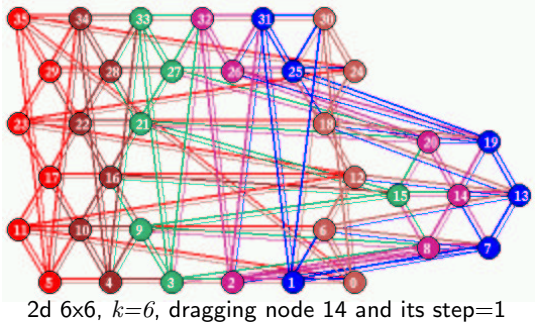
Dragging can include the node + its immediate links (step 1), the node + immediate links + their immediate links (step 2), etc. Arbitrary 1d, 2d and 3d blocks can also be dragged. Nodes with the fewest links can be automatically moved to the outer edges. This makes it easy to unravel a graph. The pre-programmed graph layouts available are a circle of nodes, a spiral, or 1d, 2d or 3d. The graph can be rotated, expanded, contracted, and various other manipulations can be performed. The graph layout can be saved/loaded. An "ant" can be launched into the network that moves according to the link probabilities (as in a Markov chain) keeping a count of node hits.



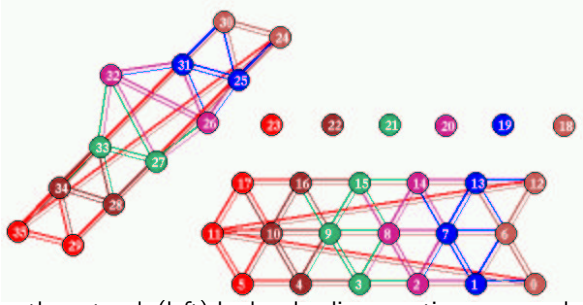
2d 8x8, k=5



3d 6x6x6, k=6, with a vertical slice (block) dragged from the center



2d 6x6, k=6, dragging node 14 and its step=1 inputs



the network (left) broken by disconnecting some nodes, a component was dragged and rotated

Figure 22: Network graphs of a 2d and 3d CA. *top left*: a 2d CA. *top right*: a 3d CA, an axonometric projection seen from below as if looking up into a cage. A vertical slice has been defined and dragged from the graph. *bottom left*: a 2d CA where the links follow a triangular grid, showing a node and its 1-step inputs dragged out, and *bottom left*: various manipulations to the graph. Note that breaking and creating new connections affects only the graph, not the underlying network which can be restored.

16 Static parameters measures

Various static parameters measures on rule look-up tables include the the λ-parameter and equivalent P-parameter, the Z-parameter, and the (weighted) average λ and Z for mixed rule networks. Also the frequency of canalizing "genes" and inputs[5, 4]. Networks can be tuned to adjust λ, Z, and canalization to any arbitrary level.

17 Measures on space-time patterns

The following measures on space-time patterns available,

- The rule-table lookup frequency histogram in a moving window of time-steps, its entropy (as in figure 3) , and the space-time pattern density.

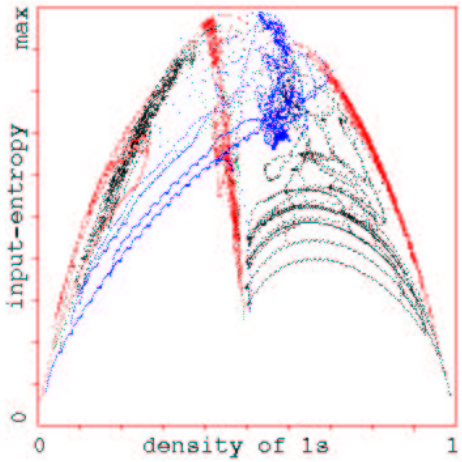


Figure 23: Entropy/density scatter plot[13]. Input-entropy is plotted against the density of 1s relative to a moving window of 10 time-steps. Plots for a number of k=5 complex rules (n=150) are show superimposed, each of which has its own distinctive signature, with a marked vertical extent, i.e. high input-entropy variance. About 1000 time-steps are plotted from several random initial states for each rule.

- An entropy/density scatter plot, where complex rules have their own distinctive signatures (figure 23).
- A scatter plot of mean entropy against the standard deviation of the entropy for an arbitrarily large sample of CA rules, which allows ordered, complex and chaotic rules to be classified automatically, also shown as a 2d frequency histogram 24. Ordered, complex and chaotic dynamics are located in different regions allowing a statistical measure of their frequency. The rules can be sorted by entropy variance allowing complex rules to be found automatically.

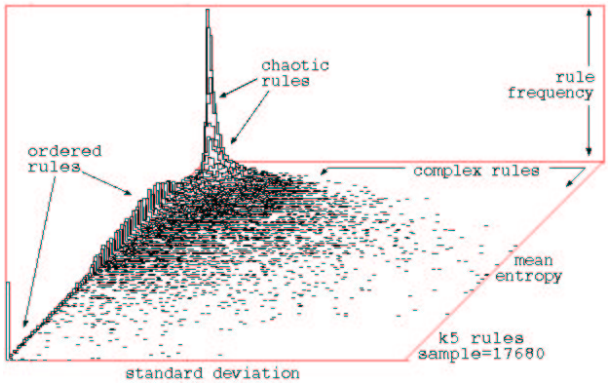


Figure 24: *left*: Classifying a random sample of k=5 rules by plotting mean entropy against the standard deviation of the entropy, with the frequency of rules within a 128x128 grid shown vertically.

- The activity/stability of network elements. Frozen islands, the fraction of "genes" that have not changed over the last x generations, the fraction of frozen 0s and 1s, and "genes" colored according the fraction of time they have been "on" (i.e. 1) in the same window of x time-steps, falling into preset "frequency bins".
- The damage spread, or pattern difference, between two networks in 1d or 2d. A histogram of damage spread frequency can be automatically generated for identical networks with initial states differing by 1 bit.
- The Derrida plot[4, 5], and Derrida coefficient, analogous to the Liapunov exponent in continuous dynamical systems, which 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.

- A scatter plot of successive iterations in a 2d phase plane, the "return map" (figure 25), which has a fractal structure, especially for chaotic rules.

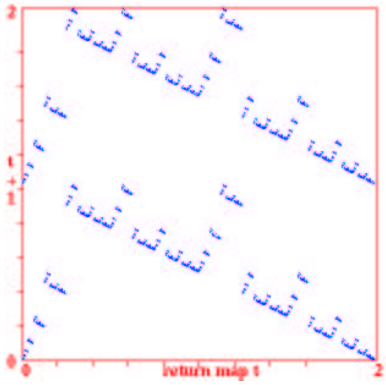


Figure 25: The return map for the $k = 3$ rule 30, n=150, for about 10,000 time-steps. Note the fractal structure. Each state (bit-string) $B_0, B_1, B_2, B_3 \ldots B_{n-1}$ is converted into a decimal number 0-2 as follows, $B_0 + B_1/2 + B_2/4 + B_3/8 + \ldots + B_{n-1}/2^{n-1}$. As the network is iterated, this value at time-step t (x-axis) is plotted against the value at timestep t+1 (y-*axis*).

18 Measures on attractor basins

The following measures on attractor basins, i.e. measures on subtrees, basins of attraction, and the basin of attraction field,

- 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.
- A histogram showing the frequency of arriving at different attractors from random initial states. This provides statistical data on the basin of attraction field for large networks. The number of basins, their relative size, period, and the average run-in length can be measures statistically. An analogous method shows the frequency of arriving at different "skeletons", partly frozen patterns.
- Garden-of-Eden density plotted against the λ and Z parameters, and against network size.
- A histogram of the in-degree frequency of attractor basins or subtrees.
- The state-space matrix, a scatter-plot of the left half against the right half of each state bit string, using color to identify different basins, or attractor cycle states.
- The attractor meta-graph (see figures 2 and 18), an analysis of the basin of attraction field tracking where all possible 1-bit flips to attractor states end up, whether to the same or to which other basin. The information is presented in two ways, as a jump-table: a matrix showing the jump probabilities between basins, and as a meta-graph: a graph with weighed vertices and edges giving a graphic representation of the jump-table. The meta-graph itself can be analyzed and manipulated in various ways, and rearranged and unraveled, including dragging vertices and defined components to new positions with "elastic band" edges; the same methods as for the network graph, section 7.

19 Reverse algorithms

There are three different reverse algorithms for generating the pre-images of a network state. These have all been generalized for multi-state networks in the next version of DDLab,

- An algorithm for 1d CA, or networks with 1d CA wiring but heterogeneous rules.
- A general algorithm for random Boolean networks, which also works for the above.
- An exhaustive algorithm that works for any "random mapping" including the two cases above.

The first two reverse algorithms 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 is largely independent of k, but is especially sensitive to network size.

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

The CA reverse algorithm applies specifically to networks with 1d CA wiring (local wiring) and homogeneous k, such as 1d CA, though the rules may be heterogeneous. This is the most efficient thus fastest algorithm, described in [9, 13]. Furthermore, compression of 1d CA attractor basins by rotation symmetry speeds up the process[9].

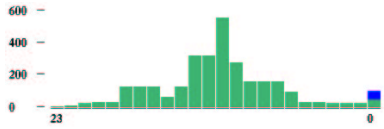


Figure 26: Computing RBN pre-images. The changing size of a typical partial pre-image stack at successive elements. n=24, k=3.

Any other network architecture, with non-local wiring, will be handled by a slower *general* reverse algorithm described in [10, 13]. A histogram revealing the inner workings of this algorithm can be displayed. Regular 2d or 3d CA will also use this general reverse algorithm though in principle more efficient algorithms that take advantage of 2d or 3d local wiring could be devised. Compression algorithms will come into play in orthogonal 2d CA to take advantage of the various rotation symmetries on the torus.

The third, brute force, reverse algorithm first sets up a mapping, a list of "exhaustive pairs", each state in state-space and its successor (this can be saved). The pre-images of states are generated by reference to this list. The exhaustive testing method is restricted to small systems because the size of the mapping increases exponentially as 2^n, and scanning the list for pre-images is slow compared to the direct reverse algorithms for CA and RBN. However, the method is not sensitive to increasing neighborhood size k, and is useful for small networks with large k. Exhaustive testing is also used for sequential updating.

A random mapping routine creates the list of exhaustive at random, possibly with some bias. The attractor basins are reconstructed by reference to this random map with the exhaustive testing algorithm. The space of random maps for a given system size corresponds to the space of all possible basin of attraction fields and is the super-set of all other deterministic discrete dynamical systems.

20 Sequential updating

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

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

21 Sculpting attractor basins

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

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 $n \leq 13$). The inverse problem is how to find a minimal network that will satisfy a full or partial mapping, fragments of attractor basins such as trajectories.

22 The multi-value version of DDLab

The multi-value version of DDLab is well advanced, and allows network elements to have a value-range v anywhere from 2 to 8. Most DDLab functions have been generalized for multi-state, including setting rules and initial states, attractor basins including compression, space-time patterns, filtering, input-entropy, the reverse algorithms for CA, RBN and random maps, and the the Z and lambda parameters. This new release will be announced in the near future.

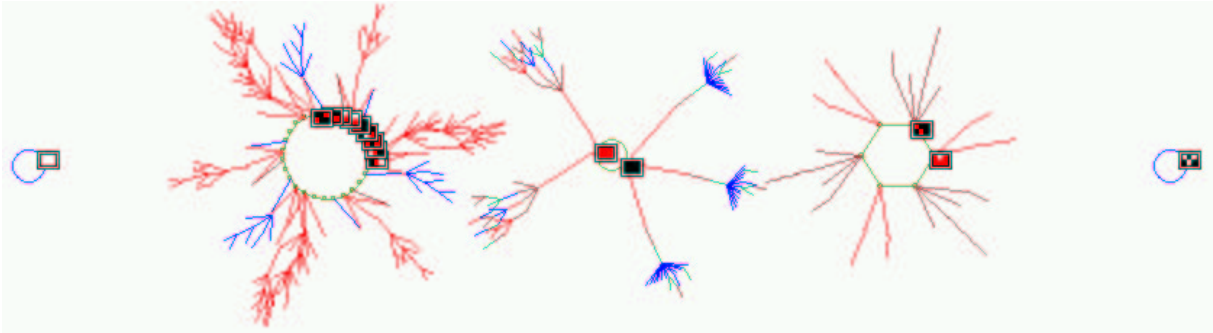


Figure 27: The basin of attraction field of a n=6, k=6 CA with a value range v=3 (i.e. values 0,1,2 colored white, red and black). The look-up table is 22221101001002122112211122 1 with a Z-parameter=0.37037. Just the 5 non-equivalent basins are shown from a total of 8, and non-equivalent attractor states are shown as a 2d patterns. State-space = 3^6 = 729.

References

[1] Adamatzky,A., "Identification of Cellular Automata", Taylor & Francis, Bristol, 1994.

[2] Albert,R., H. Jeong, and A-L Barabasi, "Error and attack tolerance in complex networks", Nature, vol 406, July 2000.

[3] Conway,J.H., "What is Life?" in "Winning ways for your mathematical plays", Berlekamp,E, J.H.Conway and R.Guy, Vol.2, chap.25, Academic Press, New York, 1982.

[4] Harris,E.S., B.K.Sawhill, A.Wuensche, and S.Kauffman, "Biased Eukaryotic Gene Regulation Rules Suggest Genome Behavior is Near Edge of Chaos", Santa Fe Institute Working Paper 97-05-039, 1997.

[5] Kauffman,S.A., "The Origins of Order", Oxford University Press, 1993.

[6] Langton,C.G., "Computation at the Edge of Chaos: Phase Transitions and Emergent Computation", Physica D, 42, 12-37, 1990.

[7] Somogyi,R., and C.Sniegoski, "Modeling the Complexity of Genetic Networks", COMPLEXITY, Vol.1/No.6, 45-63, 1996.

[8] Wolfram,S., ed. "Theory and Application of Cellular Automata", World Scientific, 1986.

[9] Wuensche,A., and M.J.Lesser. "The Global Dynamics of Cellular Automata", Santa Fe Institute Studies in the Sciences of Complexity, Addison-Wesley, Reading, MA, 1992.

[10] Wuensche,A., "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, 1994.

[11] Wuensche,A., "The Emergence of Memory", in "Towards a Science of Consciousness", (1996), eds. S.R.Hameroff, A.W.Kaszniak, A.C.Scott, MIT Press, 1996.

[12] Wuensche,A., "Genomic Regulation Modeled as a Network with Basins of Attraction", Proceedings of the 1998 Pacific Symposium on Biocomputing, World Scientific, Singapore, 1988.

[13] Wuensche,A., "Classifying Cellular Automata Automatically; Finding gliders, filtering, and relating space-time patterns, attractor basins, and the Z parameter", COMPLEXITY, Vol.4/no.3, 47-66, 1999.