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

Andrew Wuensche

Faculty of Computing, Engineering and Mathematical Sciences
University of the West of England,
and
Discrete Dynamics Inc.

1 Introduction

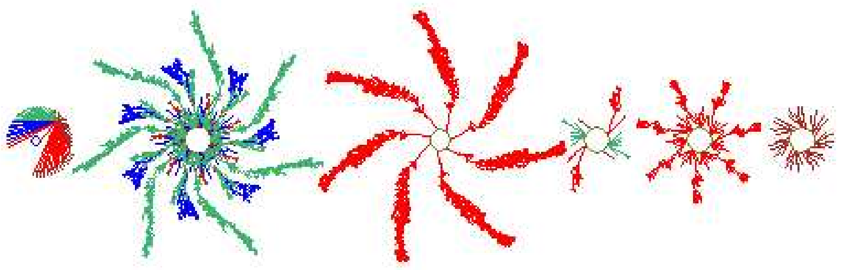


Fig. 1. The basin of attraction field of a binary (value range v=2) Cellular Automaton (CA), k=3, n=14, rule 193, with equivalent basins suppressed.

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 and economic organization; to parallel computation and artificial life; to complex systems in general.

"Decision making" networks like this are 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[4, 6, 3, 10, 12]. The networks themselves have intrinsic interest as mathematical, physical, dynamical and computational systems with a large body of literature devoted to their study[7, 8, 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.

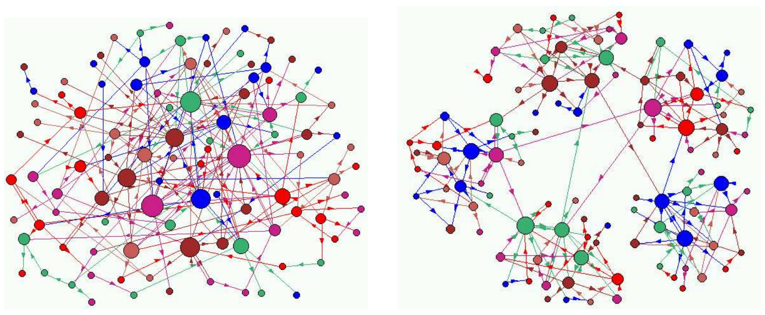


Fig. 2. Hypothetical networks of interacting elements (size n=100) with an approximate power-law distribution of connections, both inputs (k) and outputs, which are represented by directed links (with arrows). Nodes are scaled according to k and average $k \simeq 2.2$. *left*: A fully connected network.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

DDLab is able to construct these networks and investigate many aspects of their dynamical behavior. DDLab is interactive graphics software, widely used in research and education, for studying cellular automata (CA), random Boolean networks (RBN)[4] and discrete dynamical networks in general (DDN), where the "Boolean" attribute is extended to multi-value. There are currently versions of DDLab for Mac, Linux, Unix, Irix and DOS. The source code is written in C. It may be made available on request, subject to various conditions.

As well as generating space-time patterns in one, two or three dimensions, DDLab is able to construct 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. This is the network's memory, its ability to hierarchically categorize its patterns of activation (state-space), as a function of the precise network architecture[10].

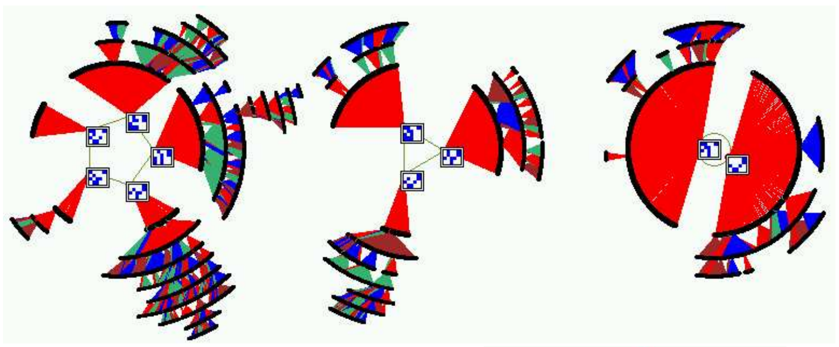
Relating this to space-time patterns in CA, high convergence implies order, low convergence implies disorder or chaos[8]. The most interesting emergent structures occur at the transition, sometimes called the "edge of chaos"[5, 13].

DDLab has recently been generalized for multi-value logic. Up to 8 values (or colors) are now possible, instead of just Boolean logic (two values - 0,1). Of course, with just 2 values selected, DDLab behaves as before[15]. Multi-values open up new possibilities for dynamical behavior and modeling.

Another major update is an option to constrain DDLab to run forward-only, to generate space-time patterns for various types of totalistic rules, reducing memory load by cutting out all basin of attraction functions. This allows larger neighborhoods (max-k=25, instead of 13). In 2d the neighborhoods are predefined to make hexagonal as well square lattices. Many interesting cellular automaton rules with "life"-like and other complex dynamics can be found in totalistic multi-value rule-space, in 3d as well as 2d[16].

DDLab is an applications program, it does not require writing code. Network parameters and the graphics presentation can be flexibly set, reviewed and altered interactively, including changes on-the-fly. There are built in tools for constructing and manipulating networks. A wide variety of measures, data, analysis and statistics are available. For small networks, its possible to compute and draw basins of attraction, and measure their convergence and stability to perturbation. For larger networks, 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 figures shown were all produced within DDLab. The operating manual[14] describes all of DDLab's many functions, and includes a "quick start" chapter. DDLab is available at `www.ddlab.com` and `www.cogs.susx.ac.uk/users/andywu/ddlab.html`.

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



	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

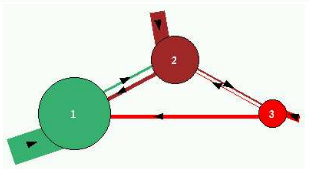
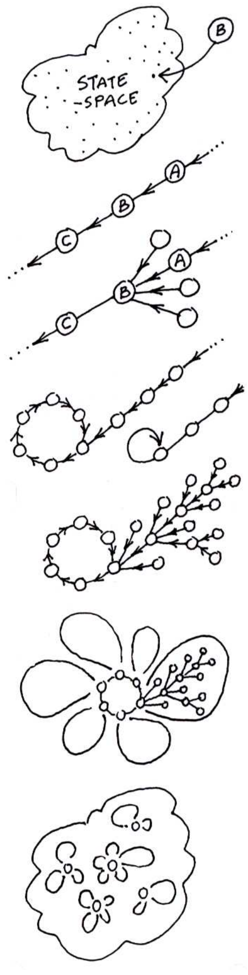


Fig. 3. The basin of attraction field of one of the n=20 sub-network shown in detail in figure 2. The binary rules were assigned at random. State-space (size $2^{20} \simeq 1.05$ million) is partitioned into three basins of attraction. The attractor states are shown as 5×4 bit patterns. The table, and diagram lower 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. All 3 basins are relatively stable because $S > V$. The lower right diagram, the "attractor jump-graph" shows the same data graphically; node size reflects basin volume, link thickness percentage jumps, arrows the direction, and the short stubs self-jumps.

2 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.



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

Part of a *trajectory* in state-space, where C is a successor of B, and A is a predecessor (*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.

Fig. 4. State-space and basins of attraction.

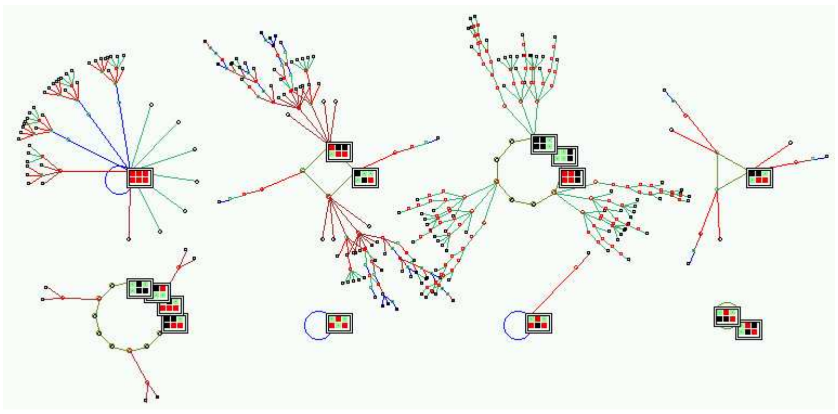


Fig. 5. The basin of attraction field of a multi-value v=3 n=6, k=3 CA. The look-up table is 120201201020211201022121111 (1886122584a655 in hex). Just the 8 non-equivalent basins are shown from a total of 23, and attractor non-equivalent states are shown as a 2d patterns. State-space $= v^k = 3^6 = 729$.

3 Discrete dynamical networks

Acronym glossary:

- CA: Cellular automata: nearest neighbor wiring and a homogeneous rule.
- RBN: random Boolean networks: random wiring and heterogeneous rules, possibly heterogeneous neighborhoods k.
- DDN: discrete dynamical networks: including RBN, but allowing a value range $v \leq 2$. CA and RBN are special cases of DDN.

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 the arriving signals. More abstractly, the network is made up of elements 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), 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, in discrete "time-steps", but may also be sequential in a predetermined order.

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

- The value-range, v. The range of values that are available to a cell. In other words, the number of possible internal states of the cell, or colors, or letters in its "alphabet". In older versions of DDLab this was limited to just 2 values (0,1), but can now be selected from 2 to 8.
- The number of network elements, the system size, n.
- How the elements are arranged in space: in a 1d, 2d or 3d lattice 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.

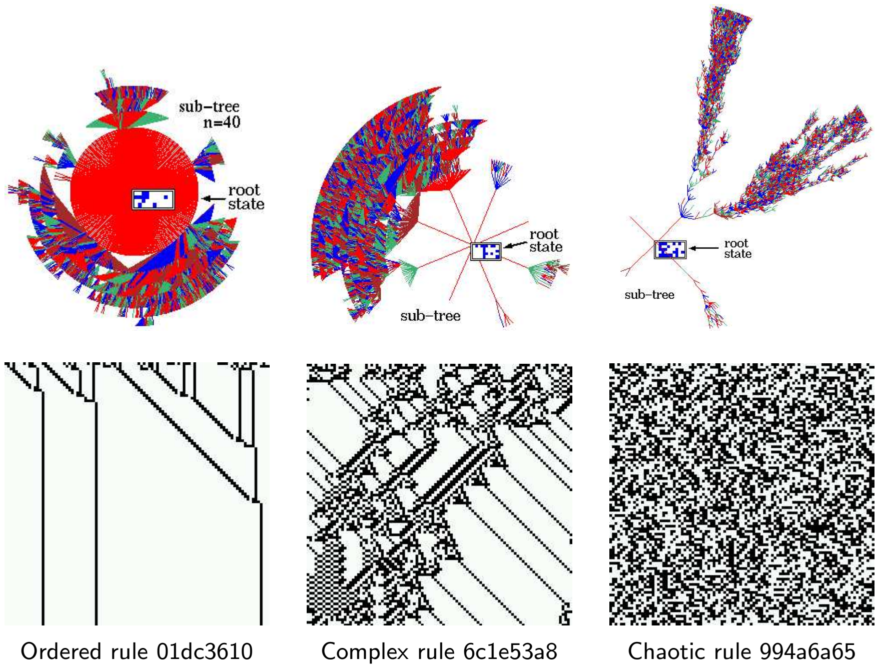


Fig. 6. Ordered, complex and chaotic dynamics of 1d binary CA are illustrated by the space-time patterns and subtrees of three typical k=5 rules (shown in hex). The bottom row shows the space-time patterns from the same random initial state. The bit-strings (n=100) of successive time-steps (represented by white and black dots) are shown horizontally one below the other; 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.

- The number of input wires, k, to each cell, or the "k-mix" if k is not homogeneous. k, may vary from 0 to 25. Maximum k is reduced for greater value-range v.
- The "wiring scheme": defining the location of the output terminals of each cell's input wires, the element's "neighborhood". CA have a homogeneous "nearest neighbor" (local) neighborhood throughout the network. RBN and DDN may have a completely arbitrary wiring scheme (a "pseudo-neighborhood"). The wiring scheme can be assigned at random, or 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 requires periodic boundary conditions, where an array's edges wrap around to their opposite edges.

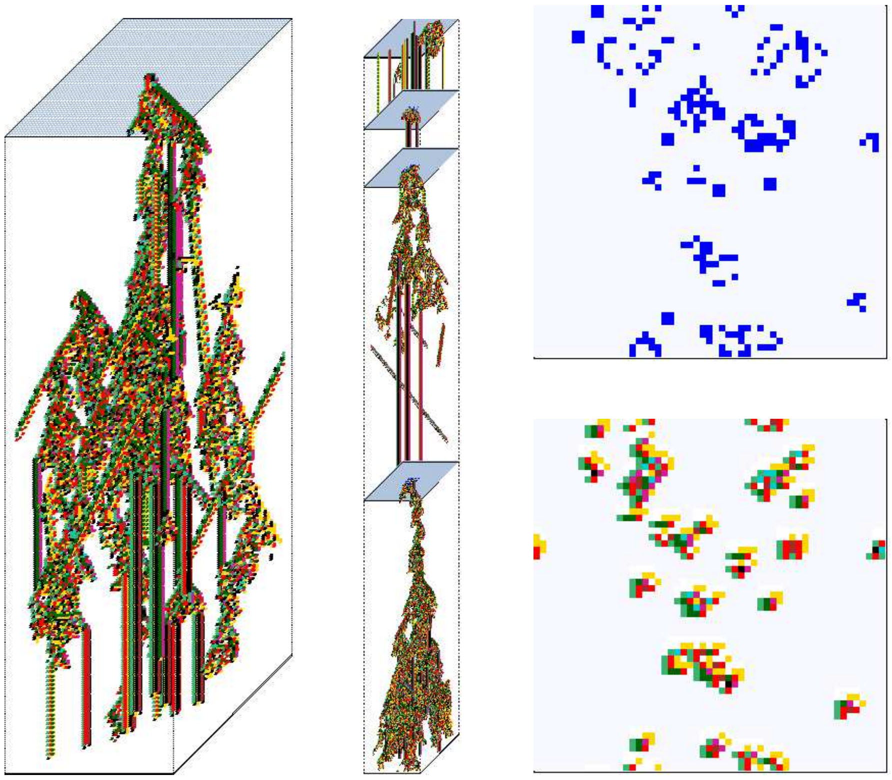


Fig. 7. Space-time pattern of the 2d game-of-Life[2], (v=2, k=9, n = 55×55) in a 3d isometric projection. 2d time-steps stack below each other, and are shown as if looking up at a transparent shaft. *left*: Starting from the "r-pentomino" seed. *center*: Re-scaled to the smallest scale, new seeds set at intervals. *upper right*: A 2d state (time-step) colored according to value. *lower right*: The same state colored according to the neighborhood look-up.

- The "rule scheme": the rules or logical functions in the network. Each element applies a rule to its inputs to compute its output. Usually this is made into a look-up table, the "rule-table", listing the outputs of all possible input patterns. CA have a homogeneous rule scheme, the same rule throughout the network. RBN and DDN may have a completely arbitrary, heterogeneous, rule scheme, or again, it may be biased in some way.

DDlab is able to create networks with any combination of these parameters, and graphically represent and analyze both the networks themselves and the dynamics resulting from the changing patterns as the complex feedback web unfolds. Network updating may be sequential as well as parallel, noisy as well as deterministic.

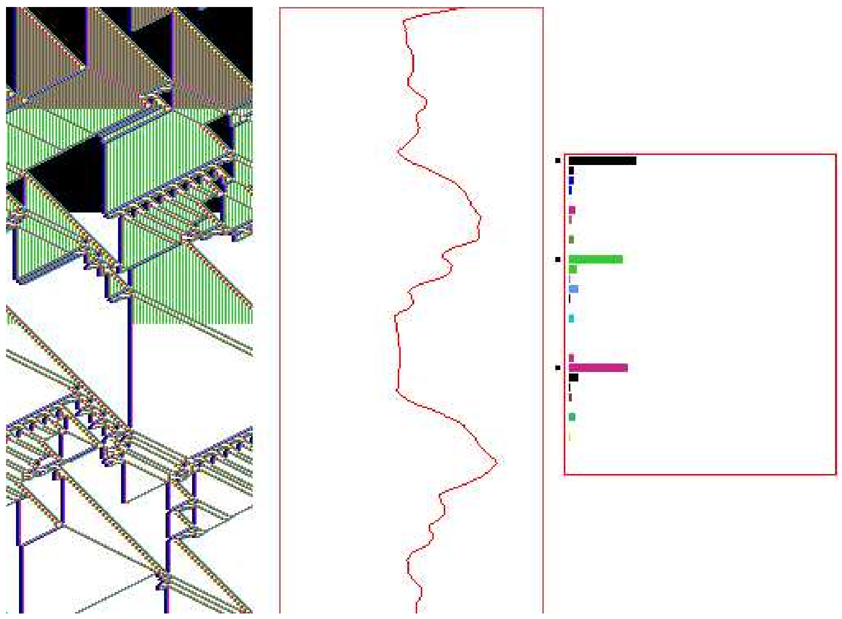


Fig. 8. A space-time pattern of a complex 1d CA, $v = 2$, $k = 5$, hex rule e9 f6 a8 15, $n = 150$. About 360 time-steps, and some analysis shown by default: *left*: The space-time pattern colored according to neighborhood look-up, and progressively "filtered" on-the-fly at three times, suppressing the background domain to show up "gliders" more clearly. *center and right*: The input-entropy plot of the lookup frequency histogram, relative to a moving window of 10 time-steps.

4 Space-time patterns and basins of attraction

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 8.

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 can be displayed as directed graphs in real time, with many presentation options, and methods

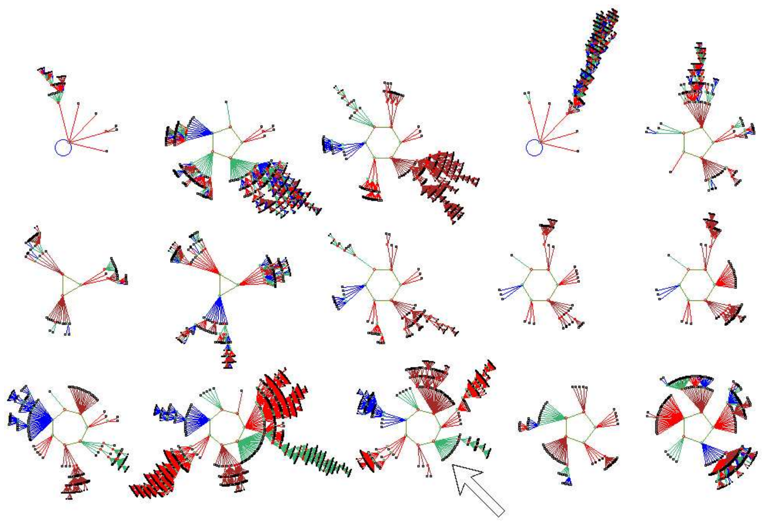


Fig. 9. The basin of attraction field of a small random Boolean network, n=13. The $2^{13} = 8192$ states in state-space are organized into 15 basins, with attractor periods ranging between 1 and 7, and basin volume between 68 and 2724. The arrow points to the basin shown in more detail.

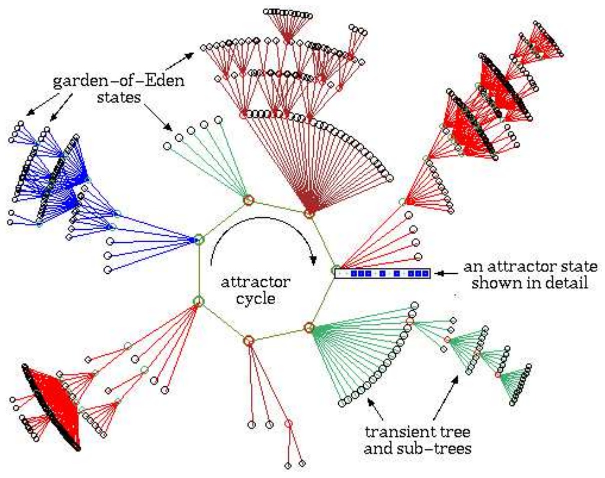


Fig. 10. One of he basins of attraction in figure 9, indicated by an arrow. The basin links 604 states, of which 523 are garden-of-Eden states. The attractor period is 7. One attractor state is shown in detail as a bit pattern. The direction of time is inwards from garden-of-Eden states to the attractor, then clock-wise

for gathering/analyzing data. The attractor basins of "random maps" may be generated, with or without some bias in the mapping.

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.

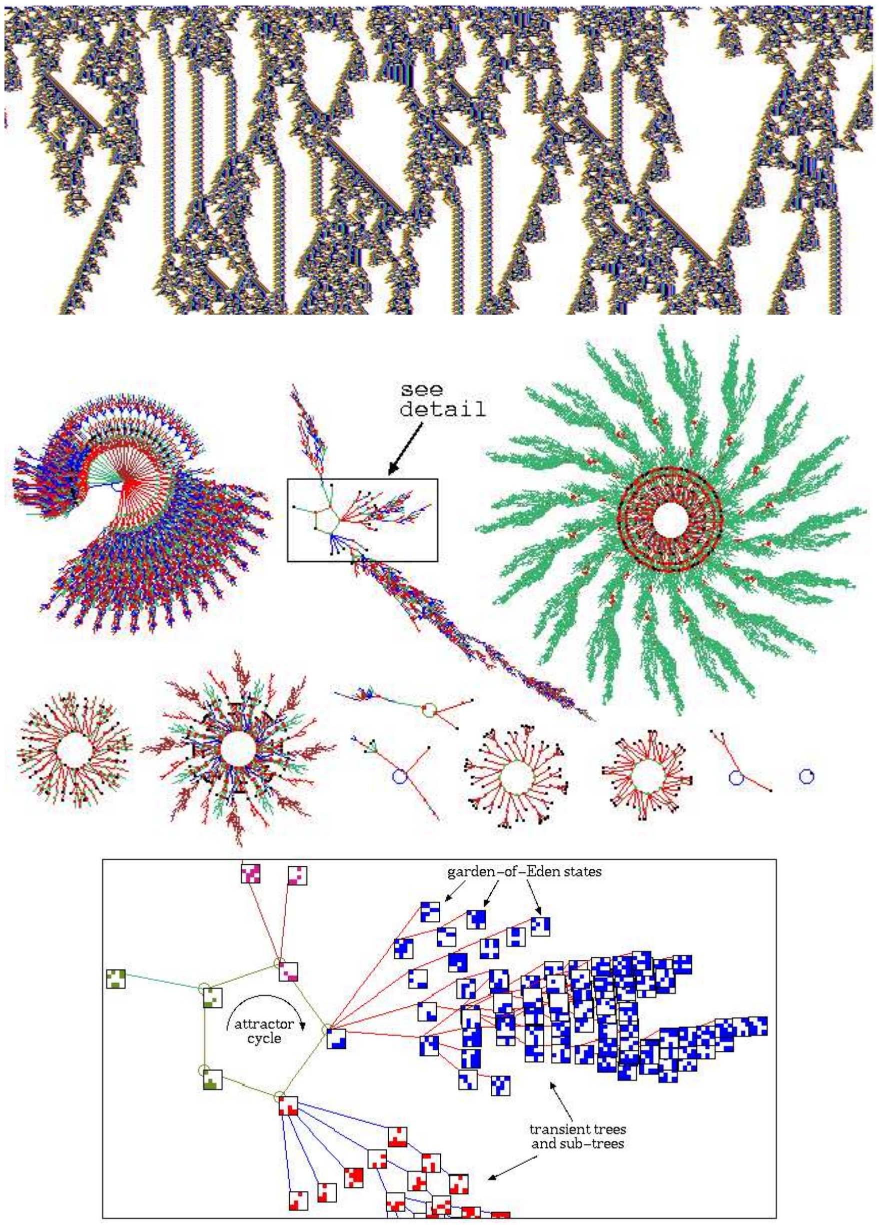
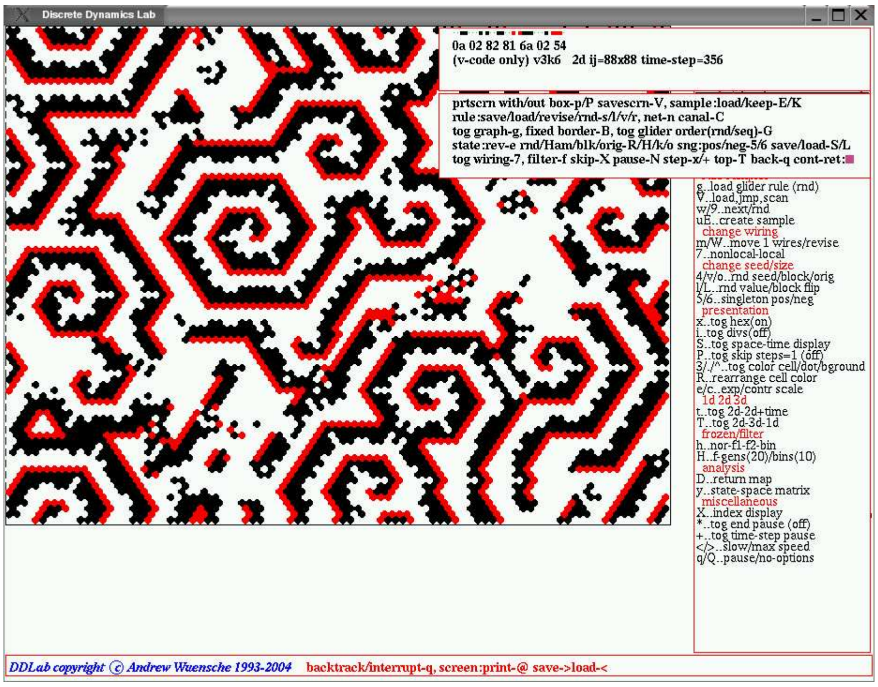


Fig. 11. *top*: The space-time pattern of a 1d complex binary CA where interacting gliders emerge[13], n=700, k=7, 308 times-steps are shown from a random initial state. *center*: The basin of attraction field for the same rule, n=16. The 2^{16} states in state-space are connected into 89 basins of attraction, 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.



```
0a 02 82 81 6a 02 54
(v-code only) v3k6   2d ij=88x88 time-step=356
```

```
prtscrn with/out box-p/P savescrn-V, sample:load/keep-E/K
rule:save/load/revise/rnd-s/l/v/r, net-n canal-C
tog graph-g, fixed border-B, tog glider order(rnd/seq)-G
state:rev-e rnd/Ham/blk/orig-R/H/k/o sng:pos/neg-5/6 save/load-S/L
tog wiring-7, filter-f skip-X pause-N step-x/+ top-T back-q cont-ret:
```

DDLab copyright © Andrew Wuensche 1993-2004 backtrack/interrupt-q, screen:print-@ save->load-<

Fig. 12. The DDLab window showing an evolving 2d CA space-time pattern, in this case on a hexagonal grid. n=88x88, v=3, k=6. The k-totalistic rule (0022000220022001122200021110, 0a0282816a0254 in hex) firstly makes gliders emerge, but spirals eventually take over. When the space-time pattern run is interrupted (with 'q'), top right windows appear giving the rule details and interrupt options; on-the-fly options are listed on the the right. A k-totalistic lookup table depends on just the frequency of the v=3 colors (2,1,0) in the k=6 neighborhood, as shown below

```
black: 2: 6 5 5 4 4 4 3 3 3 3 2 2 2 2 2 1 1 1 1 1 1 0 0 0 0 0 0 0 -
  red: 1: 0 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 - frequencies
white: 0: 0 0 1 0 1 2 0 1 2 3 0 1 2 3 4 0 1 2 3 4 5 0 1 2 3 4 5 6 -
          | | | | | | | | | | | | | | | | | | | | | | | | | | | |
          0 0 2 2 0 0 0 2 2 0 0 2 2 0 0 1 1 2 2 2 0 0 0 2 1 1 1 0 - rule table
```

5 DDLab user interface

DDLab is an interactive applications program that does not require writing code. The graphical user interface allows setting, viewing and amending network parameters, and the various presentation and analysis functions, by responding to prompts or accepting defaults.

The prompts present themselves in a main sequence for the most common 1d CA parameters. and also in a number of context dependent pop-up windows for DDN, 2d and 3d networks, and various special settings.

A flashing cursor prompts for input. Just enter **return** if in doubt, or the appropriate input from the keyboard. Press **q**, **back-space**, (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, revise, or interrupt a running process such as space-time patterns or attractor basin being generated, press **q**, or the right mouse button. To quit DDLab immediately (except for DOS) enter **Ctlr-q** at any prompt, followed by **q**. Otherwise backtrack with **q** to the start of the program.

6 Initial choices

Some initial choices in the prompt sequence set the stage for all subsequent DDLab operations, There is a choice to constrain DDLab to run forwards-only for various types of totalistic rules; this reduces memory load by cutting out full look-up tables and all attractor functions; it allows larger neighborhoods, up to max-k=25 instead of max-k=13.

If DDLab is not constrained as above there is a further choice; either to show the whole basin of attraction field, or alternatively to show something that requires an initial state: a single basin of attraction, a subtree, or just space-time patterns.

The value-range v can be set from 2 to 8. If v=2 DDLab behaves as in the old binary version. Note that as v is increased, the size of max-k will diminish, but this also depends on whether DDLab was constrained to run forwards-only for totalistic rules. For example, for v=8 and unconstrained, max-k=4 to handle the large lookup table; if constrained, max k=11.

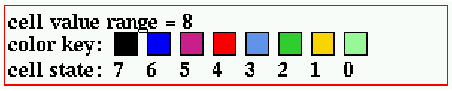

cell value range = 8
color key: ■ 🟦 🟪 🟥 🟦 🟩 🟨 🟩
cell state: 7 6 5 4 3 2 1 0

Fig. 13. The cell value color key window that appears when the value-range is selected, here for v=8. The values themselves are indexed from 7 to 0.

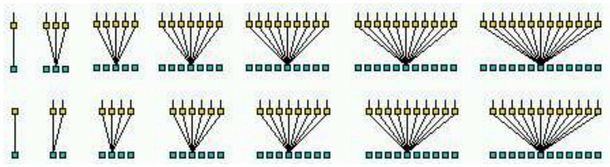
7 Setting the network size

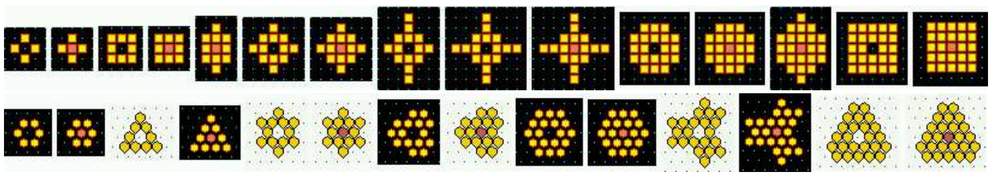
The network size n for 1d is set early on in the prompt sequence, but this is superseded if a 2d (i, j) or 3d (i, j, h) network is selected in a subsequent prompt window.

For space-time patterns, the network size is limited to n=65025, based on the maximum size of a 2d network (i, j)=255×255. 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".

For basin of attraction fields, however, the maximum network size, max-n, is much smaller, and depends on the value-range v as set out below:

v:	2	3	4	5	6	7	8
max-n:	31	20	15	13	12	11	10


1d neighborhoods: for even k the extra asymmetric cell is on the right.


2d neighborhoods k=4-25: top row square; bottom row hex; black indicates the default.

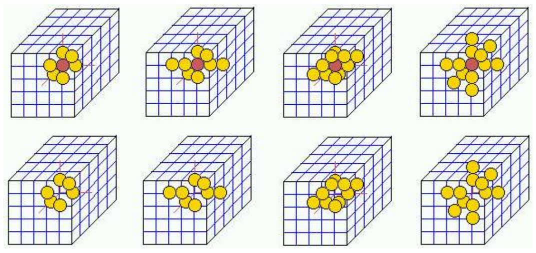

3d neighborhoods

Fig. 14. Predefined 1d, 2d and 3d neighborhoods. For 1d and 2d, $k \leq 25$ if totalistic-rules-only are set, otherwise $k \leq 13$. For 3d $k \leq 13$. For 2d the lattice/neighborhood can be either square or hexagonal.

8 The neighborhood k or k-mix

The size of the neighborhood k, the number of inputs each cell receives, can vary from 0 to max-k. Max-k itself depends on the value-range v, and also on whether or not DDLab was constrained to run forwards-only for totalistic rules. This is set out below, showing also the size of the corresponding lookup tables S.

unconstrained

v	max k	lookup S
2	13	8162
3	9	19683
4	7	16484
5	6	15629
6	5	7776
7	5	16807
8	4	4096

constrained

v	max k	lookup S
2	25	26
3	25	351
4	25	3276
5	25	23551
6	17	26334
7	13	27132
8	11	31824

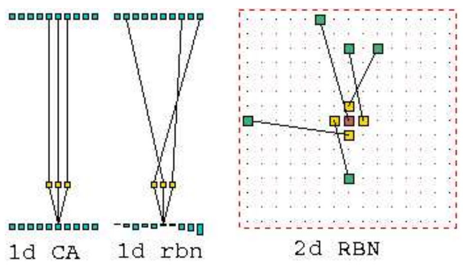
k can be homogeneous, or there can be a mix of k-values in the network. The k-mix may be set and modified in a variety of ways, including defining the proportions of different k's to be allocated at random in the network, or a "scale-free" distribution, A k-mix may be saved/loaded, but is also implicit in the wiring scheme. Figure 14 shows some predefined neighborhoods, designed to maximize symmetry. In 2d the layout can be either square or hexagonal.

9 Wiring

The network's wiring scheme, its connections, has default settings for regular CA (for 1d, 2d and 3d), with periodic boundary conditions, for each neighborhood size as shown in figure 14. Wiring can also be set at random, with a wide variety of constraints and biases, or by hand. The pre-defined neighborhoods in this case act as pseudo-neighborhoods to which the rule is applied. A wiring scheme can be set and amended just for a predefined sub-network within the network, and may be saved/loaded.

Random 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 15, in a "spread sheet" as in figure 25, or as a network graph which can be rearranged in various ways, including dragging nodes with the mouse as in figures 2 and 27.



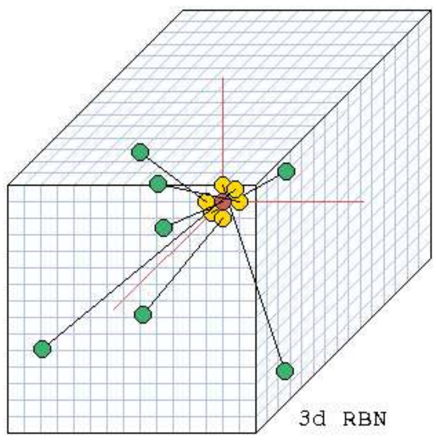


Fig. 15. RBN/DDN network wiring: cells anywhere in the network are wired back to each position in a "pseudo-neighborhood". *left:* 1d: The wiring is shown between two time-steps. *center:* 2d: k=5. *right:* 3d: k=7.

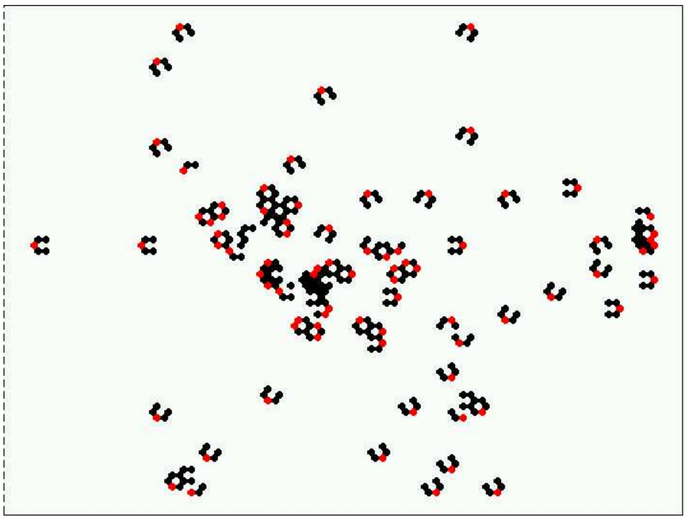


Fig. 16. A 2d CA space-time pattern, on a hexagonal grid. n=88x88, v=3, k=6. The k-totalistic rule 0022000220022001122200021210 (0a0282816a0264 in hex) allows the emergence of gliders, glider-guns and self-reproduction by glider collisions[16]. This lookup table differs by just one value from the spiral rule in figure 12

10 Rules

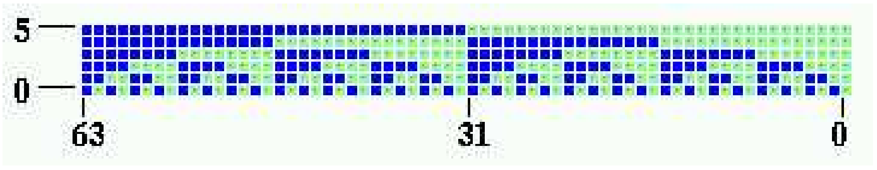


Fig. 17. The neighborhood matrix for a full lookup table for n=2 $k = 6$. All 64 possible neighborhoods from 111111 to 000000 (63 to 0) are shown vertically. The position of each neighbor is indexed 5-0. Assigning an output to each neighborhood makes the lookup table with 64 bits.

The most general update logic or rule is expressed as a full look-up table. However, there are useful subsets of the general case, two types of totalistic rules, and "outer" versions of each type. The simplest, a t-totalistic rule, depends on the sum of values in the neighborhood. k-totalistic rules depend on the frequency of each value (color) in the neighborhood (see figure12). If k=2 these two types are identical.

In addition, both types of totalistic rules can be made into outer-totalistic rules (also called semi-totalistic), where a different rule applies for each value of the central cell; the game-of-life is one such rule.

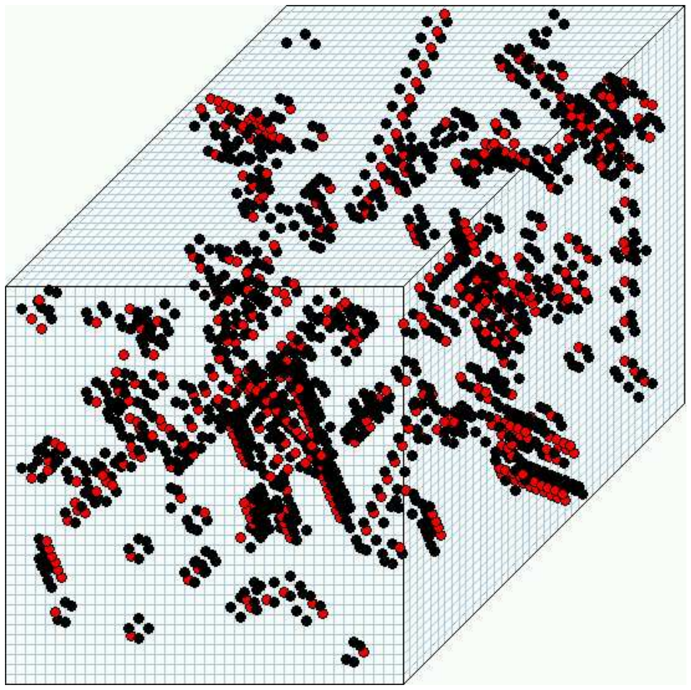


Fig. 18. A 3d CA space-time pattern, n=40×40×40. v=3, k=6 (nearest neighbors in 3d). The k-totalistic rule 0200001020100200002200120110 (200484200a0614 in hex) allows the emergence of gliders and other complex structures as in the 2d example in figure 16.

For these various types of totalistic rules, DDLab can be constrained to run forwards-only. This allows greater $[v,k]$ networks than for a full lookup table. Transformations and mutations then apply to just the constrained lookup table.

If DDLab remains unconstrained, the totalistic rules can still be selected, but they will be transformed into a full look-up table (which allows attractor basins). Transformations and mutations will then apply to this full lookup table. Within the full lookup table there are also subsets of rules that can be automatically selected at random, including symmetric rules, maximally chaotic "chain rules", Altenberg rules (figure 28), and others. The rules can be biased by various parameters, $lambda$, Z, and canalizing inputs. The "game-of-Life", "majority", and other predefined rules or rule biases can be selected.

A network may have one homogeneous rule, as for CA, or a rule-mix as for RBN and DDN. The rule-mix can be confined to a subset of pre-selected rules. Rules may be set and modified in a wide variety of ways, in decimal, hex, as a rule-table bit pattern, at random or loaded from a file. A rule scheme can be set and amended just for a predefined sub-network within the network, and may be saved/loaded..

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 (see figure 24). Rule parameters λ and Z, and the frequency of canalizing inputs in a network can be set to any arbitrary level.

11 The initial network state, the seed

An initial network state, the seed, is required to run a network forward and generate space-time patterns. A seed is also required to generate a single basin, by first running forward to find the attractor, then backward from each attractor state.

A seed is, of course, required to generate a subtree, by simply running backwards from the seed. However, for most CA rules, most states in state-space have no predecessors; they are the leaves of a subtree, "garden-of-Eden" states, so from a random seed its usually necessary to run forwards by a few steps to penetrate the subtree before running backwards, and an option is provided to do this. This was done to generate the subtrees in figure 6.

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: in decimal or hex, as a bit pattern in 1d, 2d or 3d, at random (with various constraints or biases), or loaded from a file. The bit pattern method is a mini paint program, using the keyboard to set colors (values), and the mouse or keyboard to draw those colors.

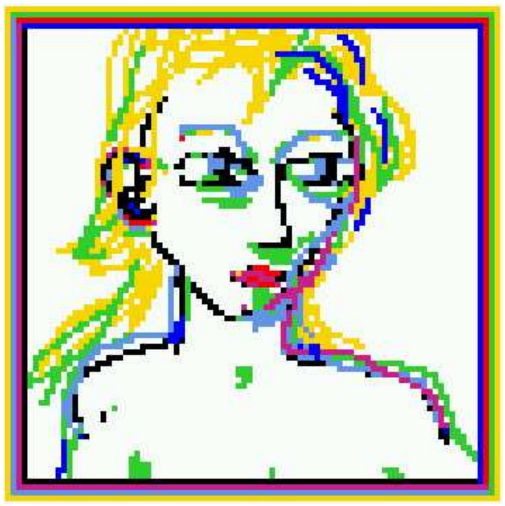
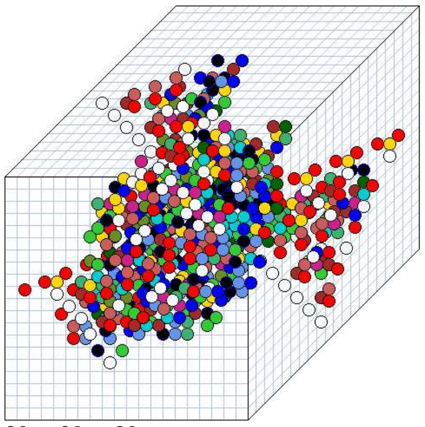


Fig. 19. Drawing a 2d initial state (seed) n=88×88, the number of colors v=8. Select the color 0 to $(v-1)$; draw with the mouse or keyboard. The image/seed can be moved, rotated and complimented. Sub-patterns saved earlier can be loaded into specified positions within the main pattern. In this example there are 8 colors. Drawing the seed also applies for 2d and 3d.



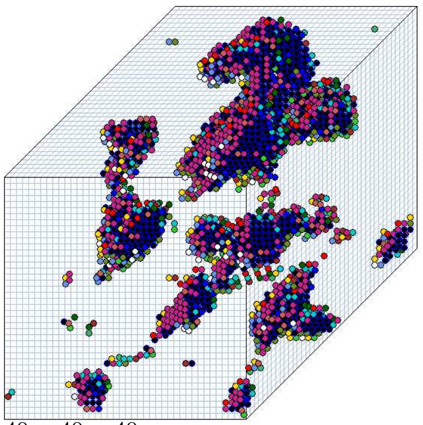


Fig. 20. Examples of 3d CA, v=2 k=7. The projection is axonometric seen from below, as if looking up at the inside of a cage. Cells are shown colored according to neighborhood lookup for a clearer picture (instead of by value: 0,1). *left*: n=20×20×20, with a randomly selected rule. The initial state is a "singleton seed", a single *on* cell in an otherwise empty array. *right*: n = 40×40×40 (the maximum size DDLab supports), The initial state was set at random, but with a bias of 45% of *on* cells.

12 Networks of sub-networks

Its possible to create a system of independent or weakly coupled sub-networks (as in figure 2), either directly, or by saving smaller networks to a file, then loading them at appropriate positions in a 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 attributes to accommodate the sub-network. For example, to load a DDN into a CA, the CA may need be set up as if it were a DDN. To load a mixed-k sub-network into single-k base network, k in the base network needs to be at least as big as the biggest k in the sub-network. Options are available to 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. There is a function to see the difference pattern (or damage spread) between two networks from similar initial states.

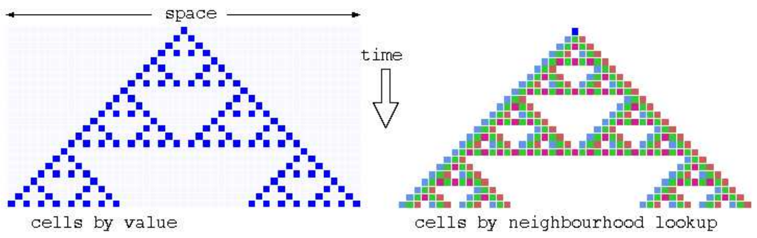


Fig. 21. Space-time patterns of a binary 1d CA (n=24, $k = 3$, rule 90). 24 time-steps from an initial state with a single central 1. Two alternative presentations are shown. *Left*, cells by value, *Right*, cells colored according to their look-up neighborhood.

13 Presentation options for space-time patterns

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

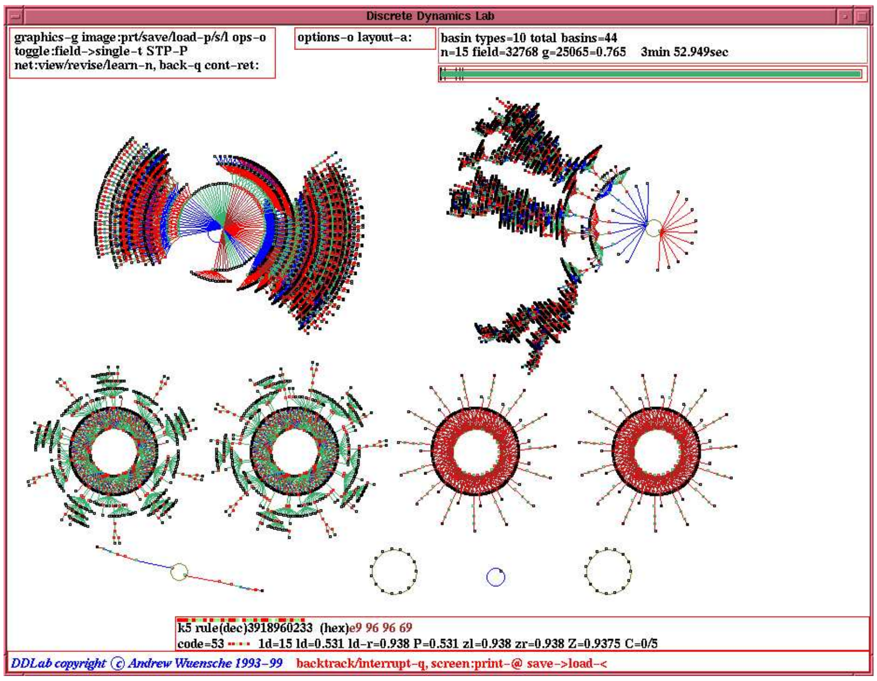
Cells in space-time patterns are colored according to their value, or alternatively according to their neighborhood at the previous time step, the entry in the look-up table that determined the cell's value. A key press will toggle between the two. Space-time patterns can be filtered to suppress cells that updated according to the most frequently occurring neighborhoods, thus exposing "gliders" and other structures, as in figure 8.

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 (order) depends on "canalizing inputs", and is applied in Kauffman's RBN model of gene regulatory networks[4, 3].

A 1d space-time pattern may be presented in successive vertical sweeps, or may be continuously scrolled. 2d networks can be toggled between square and hexagonal layout. 2d networks can also be displayed with a time dimension (2d+time) in a 3d isometric projection, as is figure 7 for the "game-of-Life". 3d networks are presented within a 3d "cage" (figures 18 and 20). 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 skipping time-steps, reversing to previous time-steps, 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. This can be rearranged in any arbitrary way, including various default layouts. For example a 1d space-time pattern can be shown in a circular layout.



```
graphics-g image:prt/save/load-p/s/l ops-o
toggle:field->single-t STP-P
net:view/revise/learn-n, back-q cont-ret:
```

```
options-o layout-a:
```

```
basin types=10 total basins=44
n=15 field=32768 g=25065=0.765    3min 52.949sec
```

```
k5 rule(dec)3918960233  (hex)e9 96 96 69
code=53        1d=15 ld=0.531 ld-r=0.938 P=0.531 zl=0.938 zr=0.938 Z=0.9375 C=0/5
```

```
DDLab copyright © Andrew Wuensche 1993-99    backtrack/interrupt-q, screen:print-@ save->load-<
```

Fig. 22. The DDLab screen showing a basin of attraction field. This example is for a binary 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.

14 Presentation options for attractor basins

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

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

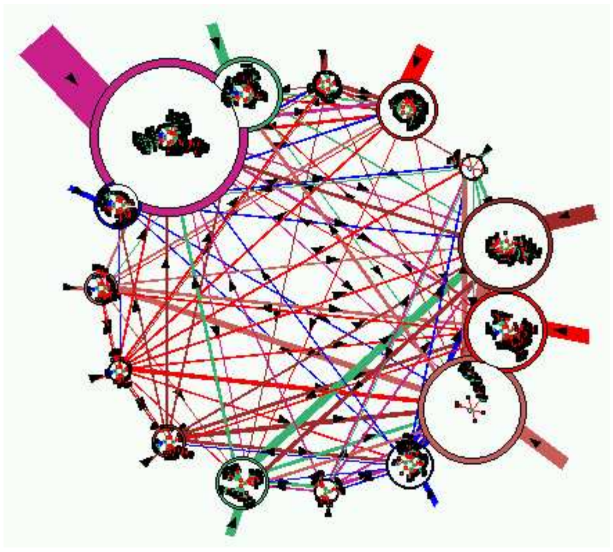


Fig. 23. The basin of attraction field (in figure 9) with each basin redrawn within the nodes of the attractor jump-graph. The jump-graph shows the probability of jumping between basins due to single bit-flips to attractor states. Nodes representing basins (shown inside each node) are scaled according 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 jump-graph itself can be suppressed, making this an alternative flexible method for positioning basins.

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.

15 Filing

DDLab allows filing a wide range of internally defined 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], as in figure 30. A large collection of complex rules, those featuring "gliders" or other large scale emergent structures, can be assembled. Pre-assembled files of CA rules sorted by this method are provided with DDLab.

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 any external screen grabber.

16 Mutations

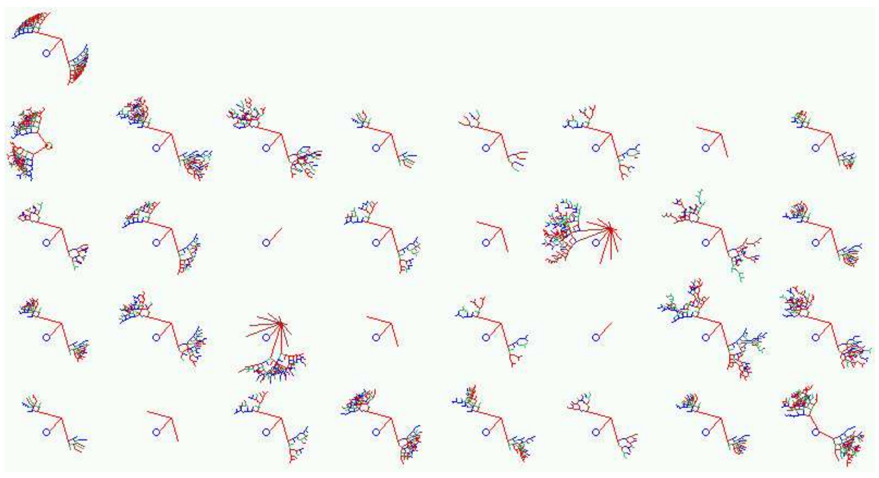
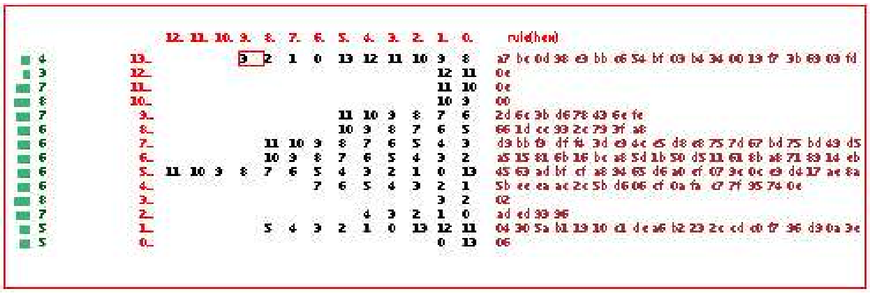


Fig. 24. 32 mutant basins of attraction of the v=*2*, k=*3* rule 195 (n=*8*, seed all 0s). *top left*: The original rule, where all states fall into just one very regular basin. The rule was first transformed to its equivalent k=*5* rule (f00ff00f in hex), with 32 bits in its rule-table for finer mutations. All 32 one-bit mutant basins are shown. If the rule is the genotype, the basin of attraction can be seen as the phenotype.

As well as on-the-fly changes to presentation, a wide variety of on-the-fly network "mutations" can be made.

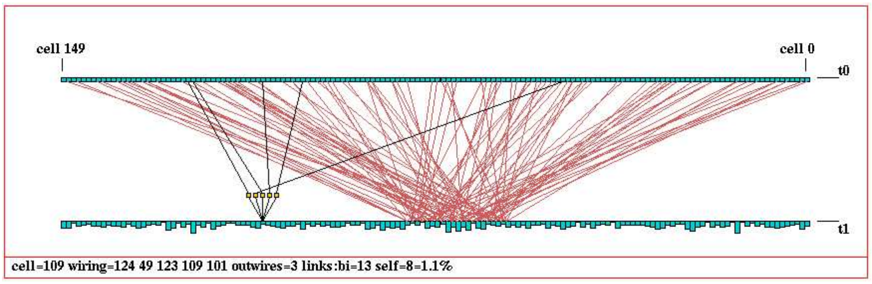
When running forward, key-press options allow mutations to wiring, rules, and current state. A number of "complex" CA rules (with glider interactions), are provided as files with DDLab, and these can be activated on-the-fly.

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 illustrated in figure 24.



		12.	11.	10.	9.	8.	7.	6.	5.	4.	3.	2.	1.	0.	rule(hex)
4	13._				9	2	1	0	13	12	11	10	9	8	a7 bc 0d 38 e3 bb c6 54 bf 03 b4 34 00 13 f7 3b 63 03 fd
3	12._												12	11	0e
7	11._												11	10	0e
8	10._												10	9	00
7	9._								11	10	9	8	7	6	2d 6c 3b d6 78 43 6e fe
6	8._								10	9	8	7	6	5	66 1d cc 93 2c 79 3f a8
6	7._					11	10	9	8	7	6	5	4	3	d9 bb f9 df f4 3d e3 4c e5 d8 e8 75 7d 67 bd 75 bd 43 d5
6	6._					10	9	8	7	6	5	4	3	2	a5 15 81 6b 16 bc a8 5d 1b 50 d5 11 61 8b a8 71 83 14 eb
6	5._	11	10	9	8	7	6	5	4	3	2	1	0	13	45 63 ad bf cf a8 34 65 d6 a0 ef 07 3c 0c e3 d4 17 ae 8a
6	4._							7	6	5	4	3	2	1	5b ee ea ac 2c 5b d6 06 cf 0a fa c7 7f 95 74 0e
8	3._												3	2	02
7	2._									4	3	2	1	0	ad ed 93 96
5	1._					5	4	3	2	1	0	13	12	11	04 30 5a b1 13 10 c1 de a6 b2 23 2c cd c0 f7 36 d9 0a 3e
5	0._												0	13	06

Fig. 25. The wiring matrix for a mixed k network with random wiring. n=14, k=2-13, with binary rules. 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.



cell=109 wiring=124 49 123 109 101 outwires=3 links:bi=13 self=8=1.1%

Fig. 26. 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.

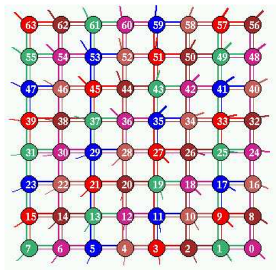
17 Network architecture

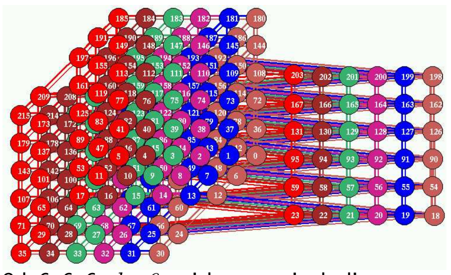
DDLab provides methods for reviewing and amending network architecture: both wiring and rules: From the wiring matrix (figure 25) and from the network architecture graphic (figure 26), which can be displayed in 1d, 2d or 3d. The network's connections and rules can be examined, changed, and tailored to requirements, including biased random settings to pre-defined parts of the network. These are very flexible methods, and for RBN/DDN its usually easier to set up a suitable dummy network initially, then tailor it here.

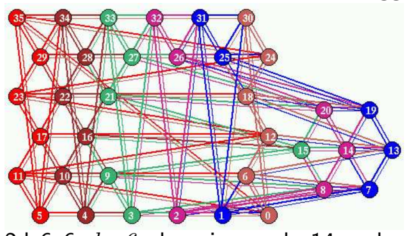
Network connectivity measures from the network architecture graphic include the following,

- 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.


2d 8x8, k=5


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


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

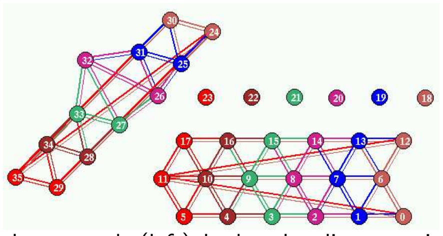

the network (left) broken by disconnecting some nodes, a component was dragged and rotated

Fig. 27. 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 hexagonal lattice, showing a node and its 1-step inputs dragged out, and *bottom right*: various manipulations to the graph. Note that breaking and creating new connections affects only the graph, not the underlying network which can be restored.

18 The network graph

Another method of reviewing network architecture is an adjacency matrix and network graph (see figures 3 and 27) that looks just at the network connections, nodes linked by directed edges. It does not allow changes to the underlying network, but includes flexible methods for representing the network, and rearranging and unraveling its graph.

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. The average directed shortest path, and non-directed small world distance can be calculated. Arbitrary 1d, 2d and 3d blocks can be dragged. Nodes with the fewest links can be automatically moved to the outer edges. This makes it possible 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.

19 Static parameters measures

Various static parameters measures on rule look-up tables include the the λ-parameter and equivalent P-parameter, the Z-parameter, which is generalized for multi value, and the (weighted) average λ and Z for mixed rule networks. The frequency of canalizing "genes" and inputs[4, 3], and Post functions.

Single rules or a rule-mix can be tuned to adjust any of these measures to any arbitrary level.

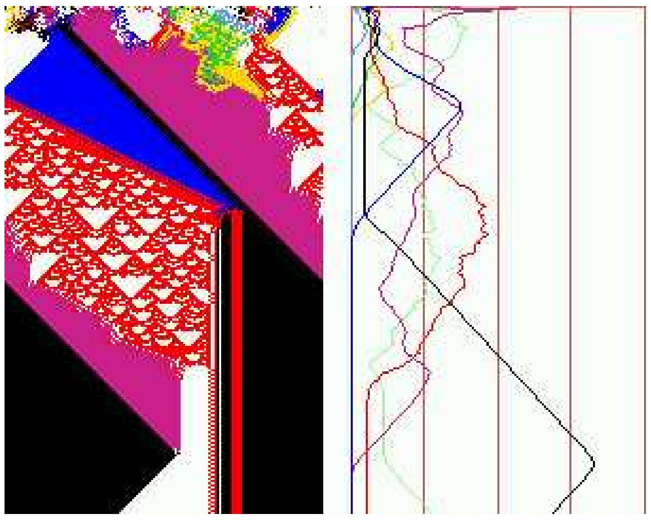


Fig. 28. A 1d CA of an Altenberg rule (v=8, k=7, n=150), where the probability of a rule-table output depends on the fraction of colors in its neighborhood. On the right the color density is plotted for each of the 8 colors, relative to a moving window of 10 time-steps.

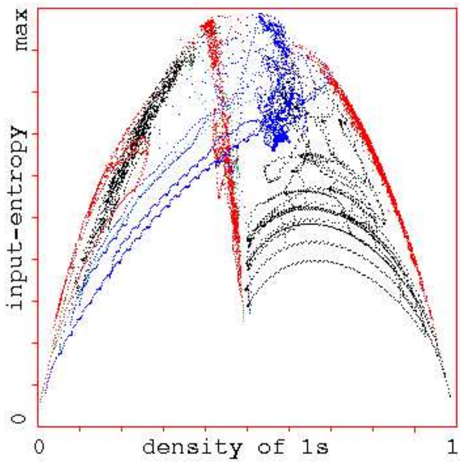


Fig. 29. Entropy/density scatter plot[13]. Input-entropy is plotted against the density of 1s relative to a moving window of 10 time-steps. Plots for a number of k=5 complex rules (n=150) are show superimposed, each of which has its own distinctive signature, with a marked vertical extent, i.e. high input-entropy variance. About 1000 time-steps are plotted from several random initial states for each rule.

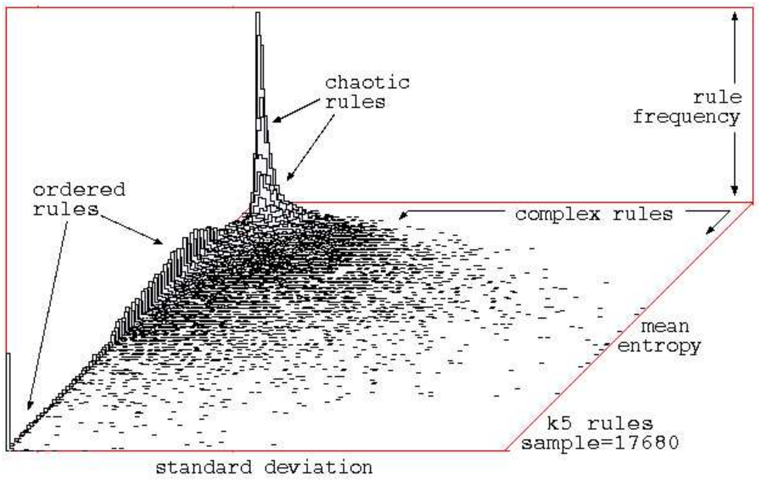


Fig. 30. Classifying a random sample k=5 rules by plotting mean entropy against the standard deviation of the entropy, with the frequency of rules within a 128x128 grid shown vertically.

20 Measures on space-time patterns

Some measures on space-time patterns are listed below:

- The rule-table lookup frequency histogram in a moving window of time-steps, and its entropyplot (figure 8). This is the basis of the method for automatically filtering space-time patterns[13] as in figure 31.
- The space-time color density in a moving window of time-steps(figure 28).

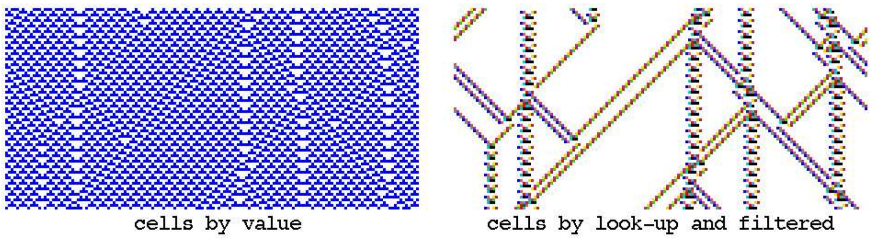


Fig. 31. Filtering a binary 1d space-time pattern with interacting gliders embedded in a complicated background *left*, and the same space-time pattern filtered *right*. Filtering is done on-the-fly for any rule. In this example, k=3 rule 54 was first transformed to its equivalent k=5 rule (hex: 0f3c0f3c). n=150.

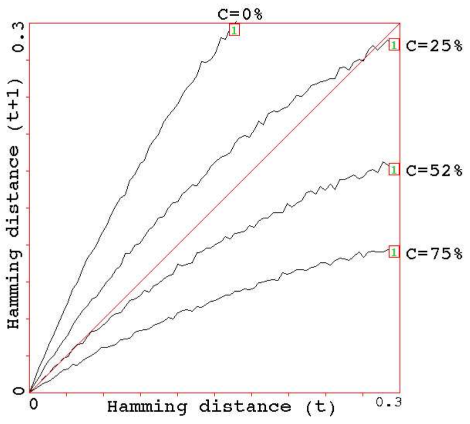


Fig. 32. Derrida plots for a random Boolean networks (36×36, k=5). This is a statistical measure of how pairs of network trajectories diverge/converge in terms of their Hamming distance. A curve above the main diagonal indicates divergence and chaos, below - convergence and order. A curve tangential to the main diagonal indicates balanced dynamics. This example shows 4 plots where the the percentage of canalizing inputs in the randomly biased network is 0%, 25%, 52%, and 75%, showing progressively greater order.

- The variance of the entropy, and an entropy/density scatter plot, where complex rules have their own distinctive signatures (figure 29).
- 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 (figure 30). 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.
- Various methods for showing the activity/stability of network elements.

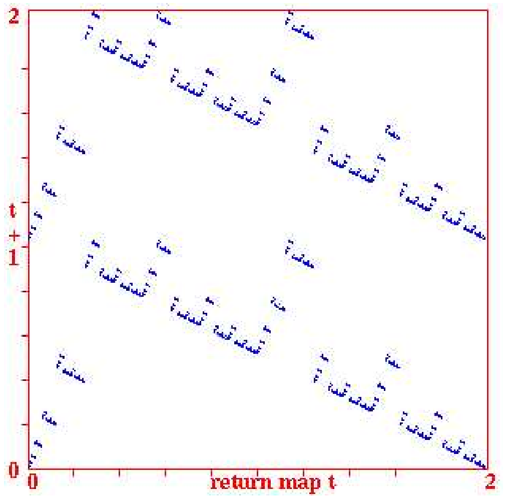


Fig. 33. The return map for binary 1d k=3 rule 30, n=150, for about 10,000 time-steps. Note the fractal structure. Each state (bitstring) $B_0, B_1, B_2, B_3 \ldots B_{n-1}$ is converted into a decimal number 0-2 as follows, $B_0 + B_1/2 + B_2/4 + B_3/8 + \ldots + B_{n-1}/2^{n-1}$. As the network is iterated, this value at time-step t (x-axis) is plotted against the value at time-step t+1 (y-$axis$).

- 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[3, 4], 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 (see figure 32), and is also generalized for multi-value.
- A scatter plot of successive iterations in a 2d phase plane, the "return map" (figure 33), which has a fractal structure, especially for chaotic rules.

21 Measures on attractor basins

Some measures on attractor basins, i.e. measures on subtrees, basins of attraction, and the basin of attraction field, are listed below:

- 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 is measured statistically. The data can be used to automatically generate an attractor jump-graph as in figures 3 and 23. 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 in attractor basins or subtrees.
- The state-space matrix, a 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 jump-graph (see figures 3 and 23): an analysis of the basin of attraction field tracking where all possible 1-bit flips (or 1-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 jump-table: a matrix showing the jump probabilities between basins, and as a jump-graph: a graph with weighed vertices and edges giving a graphic representation of the jump-table. The jump-graph itself can be analyzed and manipulated in various ways, and rearranged and unraveled, including dragging vertices and defined components to new positions with "elastic band" edges; the same methods as for the network graph, section 9.

22 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.

- An algorithm for 1d CA, or networks with 1d CA wiring but heterogeneous rules.
- A general algorithm for RBN/DDN, 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.

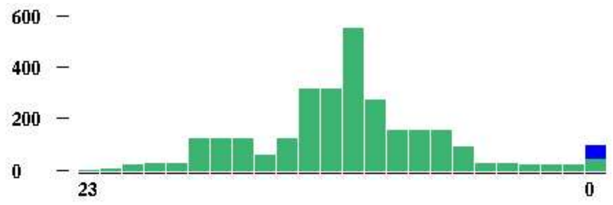


Fig. 34. Computing RBN pre-images. The changing size of a typical partial pre-image stack at successive elements. n=24, k=3. This histogram can be automatically generated for a look at the inner workings of the RBN/DDN reverse algorithm.

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 [8, 13]. Furthermore, compression of 1d CA attractor basins by rotation symmetry speeds up the process[8].

Any other network architecture, RBN or DDN, with non-local wiring, will be handled by a slower *general* reverse algorithm described in [9, 13]. A histogram revealing the inner workings of this algorithm can be displayed as in figure 34. Regular 2d or 3d CA will also use this general reverse algorithm. Compression algorithms come into play in orthogonal 2d CA to take advantage of the various rotation symmetries on the torus.

The third, brute force, exhaustive, 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 method 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 DDN. However, the method is not sensitive to increasing neighborhood size k, and is useful for small but highly connected networks. The exhaustive method is also used for sequential updating.

A random mapping routine can assign a successor to each state in state-space, possibly with some bias. Attractor basins can then be reconstructed by reference to this random map with the exhaustive 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 networks.

23 Chain rules and encryption

The CA reverse algorithm is especially efficient for a subset of maximally chaotic 1d CA rules, the "chain rules", which can be automatically generated in DDLab for any v, k. The approximate number of chain rules is $\sqrt[v]{rulespace}$.

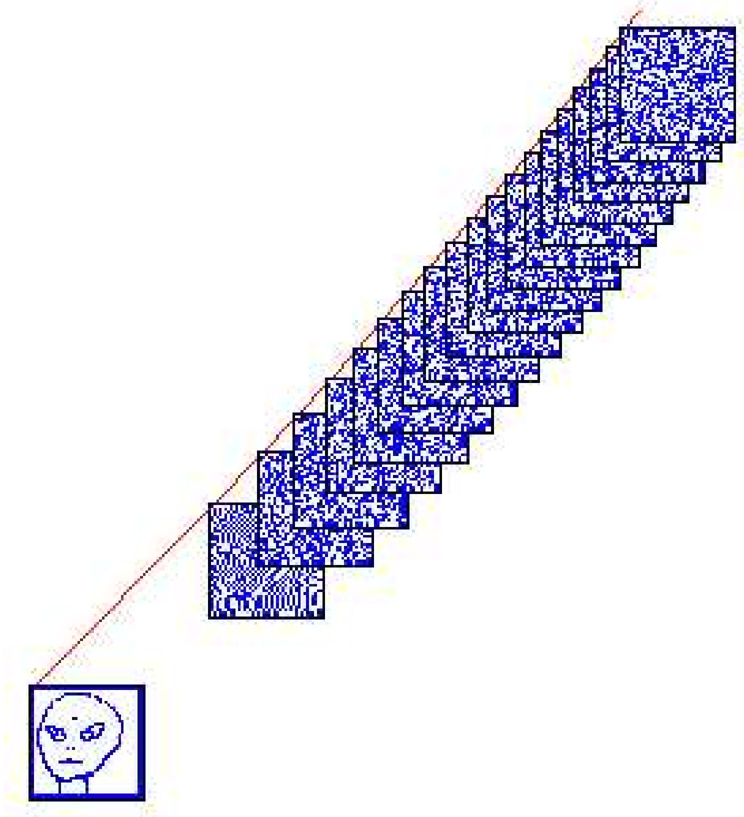


Fig. 35. A 1d pattern is displayed in 2d (n=1600, 40×40); the "alien" seed was draw as in figure 11. The seed could also be an ASCII file, or any other form of information. With a v=2, k=7 chain rule selected at random, and the alien as the root state, a subtree was generated with the CA reverse algorithm; note that the subtree has no branching, and branching is highly unlikely to occur. The subtree was set to stop after 20 backward steps which took about 12 seconds. The state reached is the encryption.

These rules are special because in contrast to the vast majority of rule-space, the typical number of predecessors of a state (in-degree) is extremely low, *decreasing* with system size. For larger systems the in-degree is likely to be exactly one. Consequently, the garden-of-Eden density is also very low and *decreasing* with system size; becoming vanishingly small in the limit. This means nearly all states have predecessors, embedded deeply along chain-like transients. Large 1d CA can be run backwards very efficiently for these rules, generating a chain of predecessors. As the rules rapidly scramble patterns, they allow a method of encryption which is available in DDLab; run backwards to encrypt, forward to decrypt (figures 35 and 36).

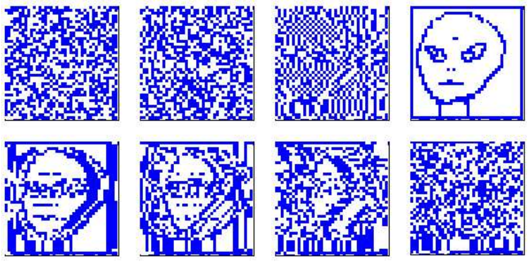


Fig. 36. To decrypt, starting from the encrypted state in figure 35, the CA with the same rule was run forward by 20 time-steps, the same number that was run backwards, to recover the original image or bit-string. This figure shows time-steps 17 to 25 to illustrate how the "alien" image was scrambled both before and after time-step 20.

24 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.

25 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.

26 Acknowledgments

DDLab has been evolving since the early 90s, mainly at the Santa Fe Institute following the publication of the "The Global Dynamics of Cellular Automata"[8] and at COGS, University of Sussex.

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 Warrel, Crayton Walker, Chris Langton, Stuart Kauffman, Wentian Li, Pedro de Oliviera, Inman Harvey, Phil Husbands, Guillaume Barreau, Josh Smith, Raja Das, Christian Reidys, Brosl Hasslacher, Steve Harris, Simon Frazer, Burt Voorhees, John Myers, Roland Somogyi, Andrew Adamatzky, Mark Tilden, Rodney Douglas, Terry Bossomaier, Ed Coxon, Oskar Itzinger, Pietro diFenizio, Pau Fernandez, Ricard Sole, Antonio Lafusa, Paolo Patelli, Jose Manuel Gomez Soto, and many other friends and colleagues (to whom I apologize for not listing). Also DDLab users who have provided feedback,

DDlab continues to be developed; updates and news can be found at `www.ddlab.com` and `www.cogs.susx.ac.uk/users/andywu/ddlab.html`.

References

1. Adamatzky,A., "Identification of Cellular Automata", Taylor & Francis, Bristol, 1994.
2. 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.
3. 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.
4. Kauffman,S.A., "The Origins of Order", Oxford University Press, 1993.
5. Langton,C.G., "Computation at the Edge of Chaos: Phase Transitions and Emergent Computation", Physica D, 42, 12-37, 1990.
6. Somogyi,R., and C.Sniegoski, "Modeling the Complexity of Genetic Networks", COMPLEXITY, Vol.1/No.6, 45-63, 1996.
7. Wolfram,S., ed. "Theory and Application of Cellular Automata", World Scientific, 1986.
8. 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.
9. 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.
10. Wuensche,A., "The Emergence of Memory", in "Towards a Science of Consciousness", eds. S.R.Hameroff, A.W.Kaszniak, A.C.Scott, MIT Press, 1996.
11. Wuensche,A., "Attractor Basins of Discrete Networks; Implications on self-organisation and memory", Cognitive Science Research Paper 461, Univ. of Sussex, D.Phil thesis, 1997.

12. Wuensche,A., “Genomic Regulation Modeled as a Network with Basins of Attraction”, Proceedings of the 1998 Pacific Symposium on Biocomputing, World Scientific, Singapore, 1998.
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.
14. Wuensche,A., “The DDLab Manual”, pdf available at www.ddlab.com.
15. Wuensche,A., “Discrete Dynamics Lab: Tools for investigating cellular automata and discrete dynamical networks”, Kybernetes, vol 32 no.1/2 2003.
16. Wuensche,A., ”Self-reproduction by glider collisions: The beehive rule”. To appear in the proceedings of ALife9, 2004.