Discrete Dynamics Lab: Tools for Investigating Cellular Automata and Discrete Dynamical Networks

Andrew Wuensche

Sept 2008

Abstract

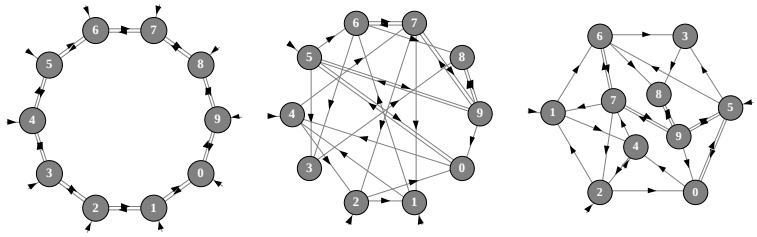
DDLab is interactive graphics software for creating, visualizing, and analyzing many aspects of Cellular Automata, Random Boolean Networks, and Discrete Dynamical Networks in general, and studying their behavior, both from the time-series perspective – space-time patterns, and from the state-space perspective – attractor basins. DDLab is relevant to research, applications and education in the fields of complexity, self-organization, emergent phenomena, chaos, collision based computing, neural networks, content addressable memory, genetic regulatory networks, dynamical encryption, generative art and music, and the study of the abstract mathematical/physical/dynamical phenomena in their own right.

1 Introduction

Networks of sparsely interconnected elements with discrete values and updating in parallel are central to our understanding of 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 of all kinds.

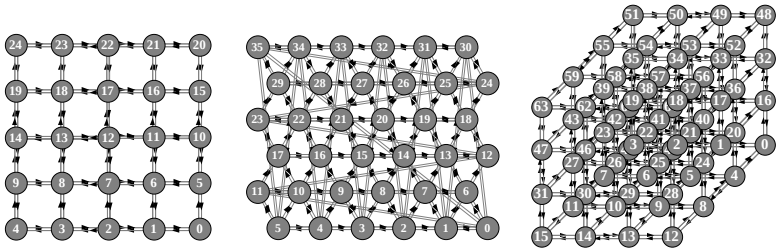
Abstract, idealized networks: Cellular Automata (CA), Random Boolean Networks (RBN), and Discrete Dynamical Networks in general (DDN), provide insights into complexity in nature by providing the simplest models of self-organization and bottom-up emergence. They are also fascinating in themselves as mathematical, physical, dynamical, and computational systems with a large body of literature devoted to their study [1, 5, 6, 8, 9, 14, 15, 16].

The dynamics that play out on these signaling – "decision-making" – discrete networks are difficult if not impossible to investigate by classical mathematics; numerical methods are therefore essential.



(a) 1D $k3$ n=10 CA (b) RBN $k3$ n=10 (c) RBN (b) unscrambled

Fig. 1: Network-graphs of: (a) The simplest 1D CA, n=10 cells and k=3 inputs per cell, one from each neighbor and one from itself. Periodic boundary conditions make the cells form a ring. Wolfram called these "elementary" CA [13]. (b) n=10, k=3 RBN wired at random, and (c) the same RBN unscrambled – nodes rearranged.



(a) 2D $k4$ n=5×5 CA (b) 2D $k6$ n=6×6 hex CA (c) 3D k=6 n=4×4×4 CA

Fig. 2: Network-graphs of: (a) A small 2D CA, (b) a 2D CA on a hexagonal lattice, (c) a 3D CA. These network-graphs can be created and manipulated in DDLab.

1.1 What is DDLab?

DDLab is interactive graphics software, able to construct, visualize and manipulate a hierarchy of discrete systems: CA, RBN, DDN (Figs. 1, 2 and 4), intermediate or hybrid networks, and random maps – and investigate and visualize many aspects of the dynamics *on* the networks, both from the time-series perspective – space-time patterns, and from the state-space perspective – attractor basins, with interactive graphical methods, as well as data gathering, analysis, and statistics. "Attractor basins" refers to any state transition graph: basin of attraction fields, single basins, or subtrees.

Fig. 3 gives a glimpse of the main themes in DDLab, and also the broad and slippery categories of the systems available:

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

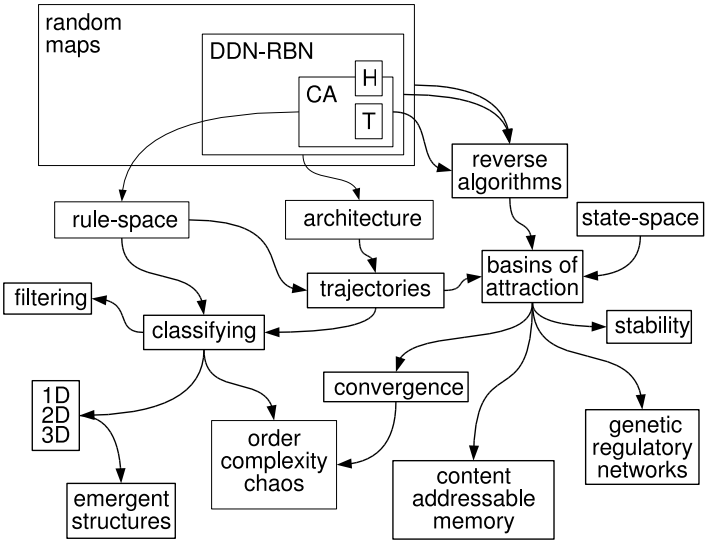
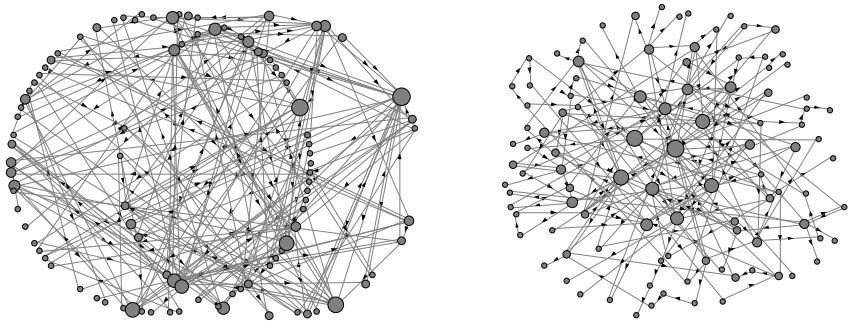


Fig. 3: The various themes, methods, functions and applications in DDLab, loosely connected. *Top Left*: The expanding hierarchy of networks: CA → RBN/DDN → within the super-set of random maps (directed graphs with out-degree one), imposing decreasing constraints on the dynamics. There are also multiple sub-categories, for example totalistic rules (T), hybrids (H) and networks of networks.

- RBN: Random Boolean Networks: random wiring of k inputs (but possibly with mixed k) and a mix of rules (but possibly one rule).
- DDN: Discrete Dynamical Networks: as RBN, but allowing a value-range $v \geq 2$. Binary CA are a special case of RBN, and RBN and multi-value CA are special cases of DDN.
- Random maps: directed graphs with out-degree one, where each state in state-space is *assigned* a successor. CA, RBN and DDN, which are usually sparsely connected ($k \ll n$) are all special cases of random maps, but if fully connected ($k = n$), rule-mix CA and the rest are equivalent to random maps.

DDLab is used widely in research and education, and has been applied to study complexity, emergence and self-organization [8, 9, 14, 16, 23, 27], in the behavior of bio-molecular networks such as neural [18, 20] and genetic [7, 11, 4, 22] networks, in many other disparate areas. As well as scientific applications, DDLab's imagery has featured in art exhibitions [24], as the light show at raves, and has been applied for generative music [2].

There are currently versions of DDLab for Mac, Linux, Unix, Irix, Cygwin and DOS. The source code is written in C – it may be made available on request subject to some conditions, though the intention is to make the code open-source in the near future.



(a) one node dragged + 2 step inputs (b) unscrambled

Fig. 4: Network-graphs showing wiring with a power-law distribution of both inputs (k=1 to 10) and outputs, n=100. Nodes are scaled by their number of inputs. (a) A circle layout, but with one node dragged together with its two-step inputs. (b) the same network unscrambled – nodes rearranged.

DDLab generates space-time patterns in one, two, or three dimensions, and also constructs attractor basins, graphs that link network states according to their transitions, analogous to Poincaré's "phase portrait" that 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 content addressable memory, its ability to hierarchically categorize all possible patterns (state-space), as a function of the precise network architecture [18, 20]. Relating this to space-time patterns in CA, high convergence implies order, low convergence implies disorder or chaos [16]. The most interesting emergent structures occur at the transition, sometimes called the "edge of chaos" [9, 23].

DDLab is generalized for multi-value logic. Up to eight values (or colors) are available, instead of just Boolean logic (two values – 0,1) in early versions. Of course, with just two values selected, DDLab behaves like the older versions. Multi-values open up new possibilities for dynamical behavior and modeling, for example fractional gene activation and reaction-diffusion.

Although DDLab is designed to run CA/DDN both forward (space-time patterns) and backwards (attractor basins), it can be constrained to run forward-only for various types of totalistic and outer-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 as 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 [26].

DDLab is an applications program, allowing network parameters and the graphics presentation to be flexibly set, reviewed and altered interactively, including changes on-the-fly. This chapter provides a brief of history, and

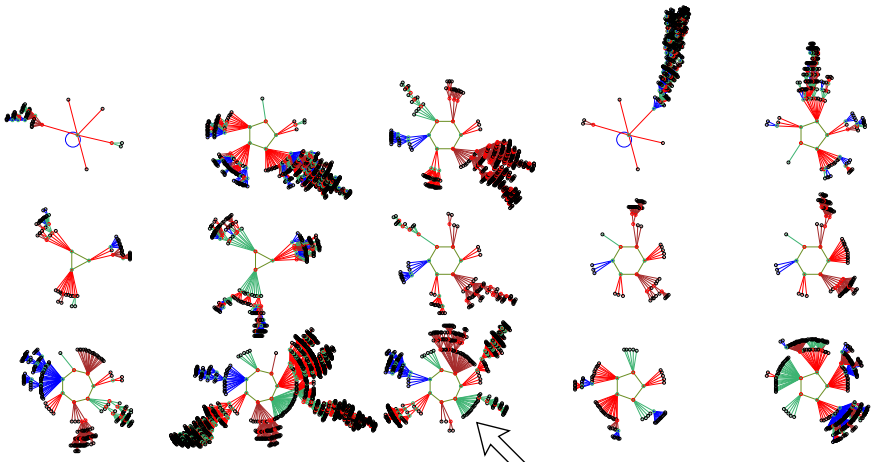


Fig. 5: The basin of attraction field of a small random Boolean network, k=3, n=13. The 2^{13} = 8192 states in state-space are organized into 15 basins, with attractor periods ranging between 1 and 7, and basin volume between 68 and 2724 [1.2sec]. The arrow points to the basin shown in more detail in Fig. 6.

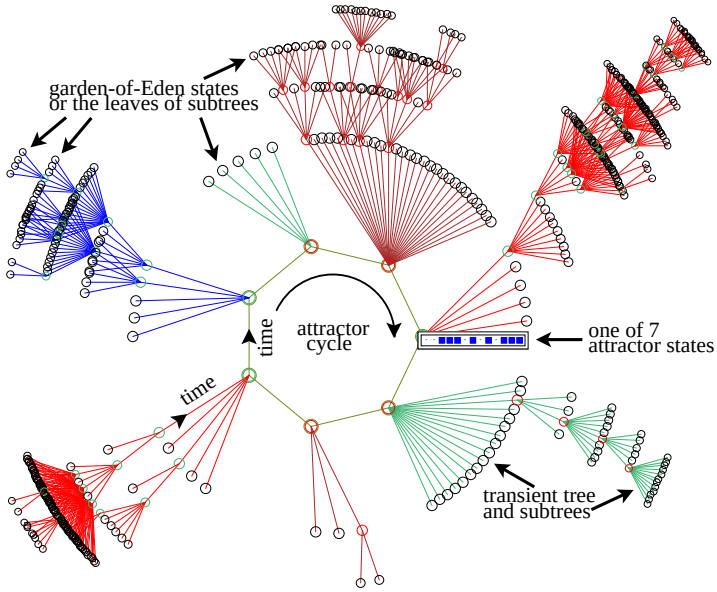


Fig. 6: A basin of attraction (one of 15) of the RBN (n=13, k=3) shown in figure 5. The basin links 604 states, of which 523 are garden-of-Eden states. The attractor period = 7, and one of the attractor states is shown in detail as a bit pattern. The direction of time is inwards from garden-of-Eden states to the attractor, then clock-wise. [0.56sec]

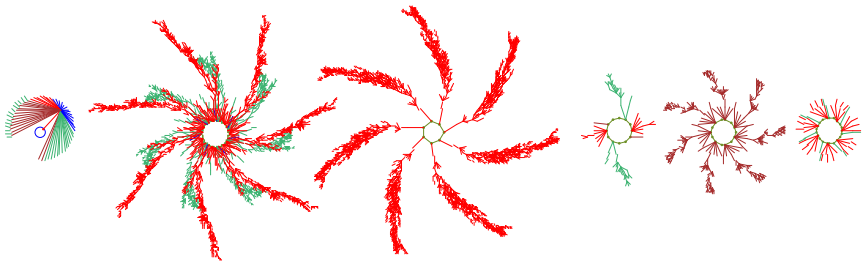


Fig. 7: The basin of attraction field of a binary (v=2) CA, neighborhood k=3, lattice size n=14 [0.2 sec]. There are in fact 15 basins, but equivalent basins have been suppressed leaving just the 6 prototypes – note also the equivalent trees typical of CA [16] but not in RBN. State nodes have been omitted. This is the famous "elementary" rule 110 [13], which is computationally universal.

gives the flavor of DDLab with a range of examples. The figures presented are drawn with various DDLab functions, including vector PostScript. The operating manual [25] (a new update is in progress) describes all of the many functions and includes a "quick start" chapter. DDLab, together with many examples, publications, and all other information, is available in [25].

DDLab remains free shareware for personal, non-commercial, users. Any other users, including commercial users, companies, government agencies, research or educational institutions, must register for a license [25].

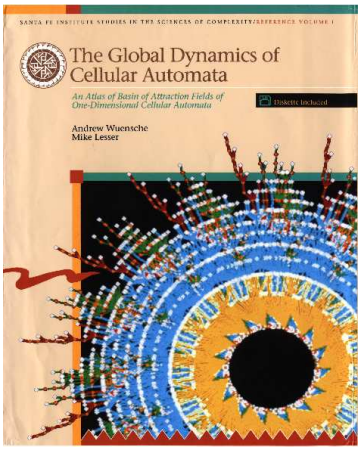
1.2 A Brief History

Two friends in London in the late 1980's became interested in CA – as to why, that is another story. They were not academic scientists. The only paper they had read on the subject was "Statistical Mechanics of Cellular Automata" [13], the first of Wolfram's many influential papers [14].

The question soon arose: can CA be run backwards as well as forwards? Not knowing that this was supposed to be impossible, an algorithm was invented the next day, and working on an Apple2 – computing a state's predecessors for rule 150. Extending the algorithm for any rule took more time and effort. Basins of attraction were computed and painfully drawn by hand.

This eventually led to the publication of their book in 1992, in the Santa Fe Institute series in the sciences of complexity: "The Global Dynamics of Cellular Automata," subtitled "An Atlas of Basin of Attraction Fields of One-Dimensional Cellular Automata" [16]. A floppy disk was attached inside the back cover with DOS software for drawing space-time patterns and basins of attraction, – the origin of what later became DDLab.

The book introduced a reverse algorithm for finding the pre-images (predecessors) of states for any finite 1D binary CA with periodic boundary



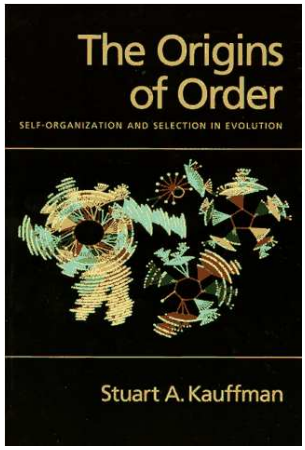


Fig. 8: The front covers of Wuensche and Lesser's (1992) "The Global Dynamics of Cellular Automata" [16] and Kauffman's (1993) "The Origins of Order" [6]. The attractor basins, of a CA and RBN, where computed and drawn with the precursor of DDLab.

.

conditions, making it possible to reveal the precise nature of their "basins of attraction" – represented by state transition graphs – states linked into trees rooted on attractor cycles, which could be drawn automatically. CA could henceforth be studied from a state-space perspective, as well as from a time-series perspective.

Two important personalities at the Santa Fe Institute (SFI) at the time were Stuart Kauffman and Chris Langton. Stuart Kauffman was famous for his Random Boolean Network (RBN) model of genetic regulation [5, 6], where cell types are seen as attractors in the dynamics on the genetic network. A meeting was arranged in the summer of 1990 in his office at the Department of Biochemistry and Biophysics, University of Pennsylvania – "OK, you've got 2 minutes" – but he was amazed to see the hand draw "atlas" of basins of attraction. Professor Kauffman ordered: "take the next flight to Santa Fe."

A long relationship with SFI began, also with Chris Langton, founder of Artificial Life which sprang from his important early CA work [8, 9]. Langton wrote the preface [10] to "The Global Dynamics of Cellular Automata," comparing the work with Poincaré, and promoted its publication. Kauffman later wrote a generous review [7].

I continued my research with the new methods, gaining new insights [17, 18, 20]. As to the software, the next task I set myself was to find an reverse algorithm for Kauffman's RBN; the CA reverse algorithm could cope with different rules at each cell, but for inputs that could arrive from anywhere – random wiring instead of wiring from a 1D CA neighborhood – that required a completely different algorithm. I invented this algorithm in 1992, in time to

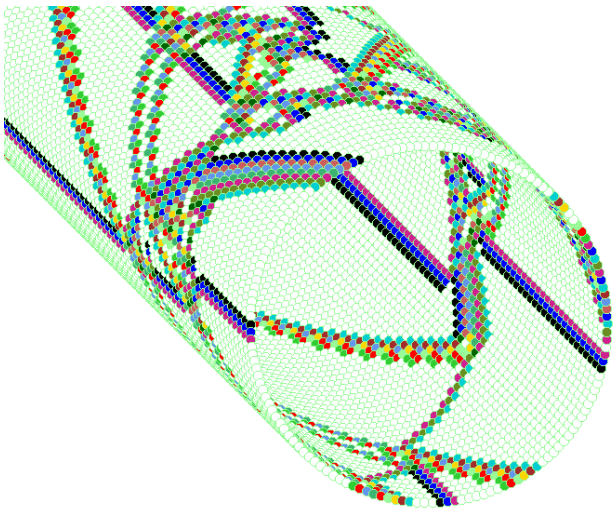


Fig. 9: A 1D space-time pattern shown as a ring of cells. scrolling diagonally towards the top left. The present moment is the ring at the bottom right. The space-time pattern is colored according to neighborhood, and has been filtered. 1D binary CA, k=5, hex rule e9 f6 a8 15, n=150 (as in Fig.16).

provide Kauffman with the cover image for his "Origins of Order" [6] (Fig. 8), and to present my results [18] at the Artificial Life III conference in Santa Fe in 1992.

In 1992 I had also started a doctorate at COGS, University of Sussex (completed in 1997 [21]) but by 1995 I was settled in Santa Fe and working at the Santa Fe Institute – in Langton's group. They where starting the Swarm project, whereas I focused on porting the DOS version of DDLab to Unix, then to Linux, with some help from the Swarm team. I named the software "Discrete Dynamics Lab" in 1993 – the first DDLab website appeared in 1996 on ALife Online, then was hosted by SFI itself in 1997.

I collaborated a great deal with Kauffman and his colleagues [4] in theoretical biology – and do still. It was at his insistence that I added a number of new RBN features to DDLab, for example: damage-spread, canalizing functions, the Derrida plot, and the attractor and skeleton frequency histograms.

1.3 DDLab Updates

DDlab has continued to grow and evolve, adding new ideas, features and platforms. Multi-value logic was added in 2003, and recently, vector PostScript for most graphic output. Below is a summary of the most significant updates with official releases on the DDLab website.

- 1995: although still in DOS, this version already had many of the im-

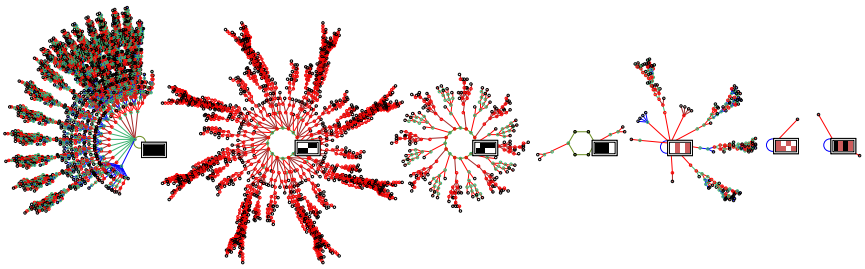


Fig. 10: basin of attraction field of a 3-value Cellular Automaton, $[v, k, n]$=[3,3,8], rule 02012012202120120101101 1022. There are in fact 17 basins, but equivalent basins have been suppressed leaving just the 7 prototypes. One attractor state is shown for each basin [0.16sec].

portant DDLab functions, described in a 63 page manual [19], CA and RBN, 1D, 2D and 2D+time space-time patterns, cell color by neighborhood lookup or frozen, look-up frequency and entropy, subtrees, basins of attraction and the basin of attraction field (for a range of network size), reverse algorithms for CA, RBN and random graphs, in-degree frequency, 1D and 2D wiring graphic and spread-sheet to review and amend the network, garden-of-Eden density, mutation, state-space matrix, learning and forgetting algorithms, filing for the network – rules – wiring.

- 1997: Unix and Linux added; sequential updating, noise, hypercube network, categorizing rule-space, attractor and skeleton frequency histograms for large networks.

- 1999: 3D networks and neighborhoods, biases for random wiring, untangling wiring, effective neighborhoods, inverse problem – pruning a fully connected network, wiring – degrees of separation, setting rules and wiring for parts of the network – layers – rows – range, filtering, value frequency color code.

- 2002: Irix added; New Manual (421 pages, 209 figures), network-graph and attractor jump-graph – where nodes can be dragged, rearranged or unravelled and space-time patterns run on the graph, 1D circle wiring graphic, wiring – power law distribution, chain-rules for encryption, Derrida coefficient, fractal return-map.

- 2003: Mac added; All aspects of DDLab generalized for multi-value logic (up to 8 values or colors), instead of just Boolean logic (0,1). Option to constrain DDLab to run forward only for totalistic rules, allowing larger neighborhoods (up to k=25), hexagonal lattices and

neighborhoods, unslanting 1D space-time patterns, 2D and 3D falling-of-edge boundary conditions, Post functions.

- 2005: Cygwin added; outer-totalistic rules, reaction-Diffusion dynamics, diagonal scrolling of 1D circle and 2D space-time patterns.

- 2008: A layout-graph for the basin of attraction field allowing basins to be dragged/rearranged into any position. Vector PostScript for most DDLab graphic output: space-time patterns, attractor basins, network-graph, jump-graph and layout-graph. Partial-order updating. Many other significant improvements. In the pipeline: the updated Manual for the latest version of DDLab.

1.4 Basins of Attraction

In continuous dynamical systems, all possible time-series make up the vector field, represented by the system's phase portrait. This is the the field of flow imposed on phase-space by the system's dynamical rule. A set of attractors, be they fixed point, limit cycles, or chaotic, attract various regions of the vector field, basins of attraction (paraphrasing Langton [9]).

Analogous concepts and terminology apply to discrete networks – though an important difference is that transients merge outside the attractor, within trajectories, far from equilibrium [20] – in continuous systems they cannot.

1.5 Times for Attractor Basins

The elapsed times to generate attractor basins are displayed in DDLab. To give an idea of these times, they are shown in the relevant figure captions in square brackets, i.e [30.1 sec] (1.66GHz Laptop running Linux/Ubuntu).

1.6 Forwards and Backwards in Time

DDLab looks at the dynamics on networks in two ways:

- from a time-series perspective, *local* dynamics, running forward.

- from a state-space perspective, *global* dynamics, running backwards.

Running forward in time 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 dynamics, including "filtering" to show up emergent structures more clearly as in Fig. 16.

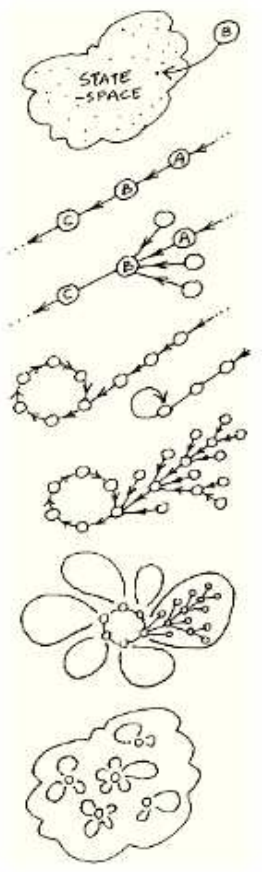
Running backwards in time generates multiple predecessors rather than a trajectory of unique successors. This procedure reconstructs the branching subtree of ancestor patterns rooted on a particular state. States without

predecessors are disclosed, the so-called garden-of-Eden states, the leaves of the subtrees.

Subtrees, 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 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 subtree by its root [18, 20]. Learning/forgetting algorithms allow attaching/detaching sets of states as predecessors of a given state by automatically mutating rules or changing connections.

Fig. 11 provides a summary of the idea of how state-space is organized into attractors and basins of attraction in discrete systems, where there is one future but many pasts.



For a binary network size n, a state B might be $1010\ldots0110$ – there are 2^n states in *State-space* (v^n for multi-value)

A *trajectory* in state-space, $A \rightarrow B \rightarrow C$

B may have other *pre-images* besides A (the *in-degree*), which can be directly computed with reverse algorithms. States with zero pre-images (leaves) are known as *garden-of-Eden* states.

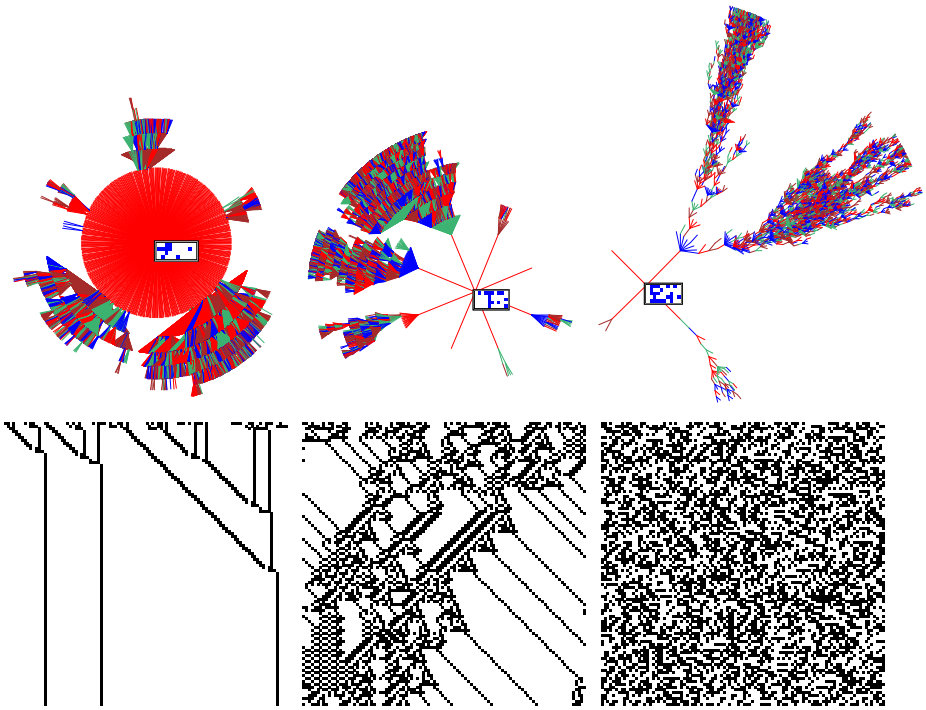
The trajectory must arrive at an *attractor*, a cycle of states with period one or more.

Find the pre-images of an attractor state (excluding the one on the attractor) – then the pre-images of the pre-images until all leaf states have been reached – the graph of linked states is the *transient tree* – 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*.

Find every attractor and construct its basin of attraction – this is the *basin of attraction field* – all states in state-space linked by the dynamics – each discrete dynamical network imposes a specific basin of attraction field on state-space.

Fig. 11: State-space and basins of attraction – the idea – one future, many pasts.



(a) Ordered rule 01dc3610 (b) Complex rule 6c1e53a8 (c) Chaotic rule 994a6a65

Fig. 12: 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 squares) are shown horizontally one below the other; time proceeds down. Above each space-time pattern is a typical subtree for the corresponding rule. The 1D subtree roots are shown in 2D: n=40 (4×10) for the ordered rule, and n=50 (5×10) 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 backward. The ordered subtree is complete; the complex and chaotic subtrees were stopped after 12 and 50 backward time-steps. Note that the convergence in the subtrees, their branchiness or typical in-degree, relates to order-chaos in space-time patterns, where order has high, chaos low, convergence. [(a) 30.1sec, (b) 42.3sec, (c) 5.8sec]

2 DDLab's interface and Initial Choices

DDLab's graphical user interface allows setting, viewing, and amending network parameters, and provides presentation, output and analysis options, by responding to prompts or accepting defaults. Navigating the interface takes a bit of practice, but it is designed to be both powerful and flexible, bypassing the more specialized options unless required, with CA the simplest to set up.

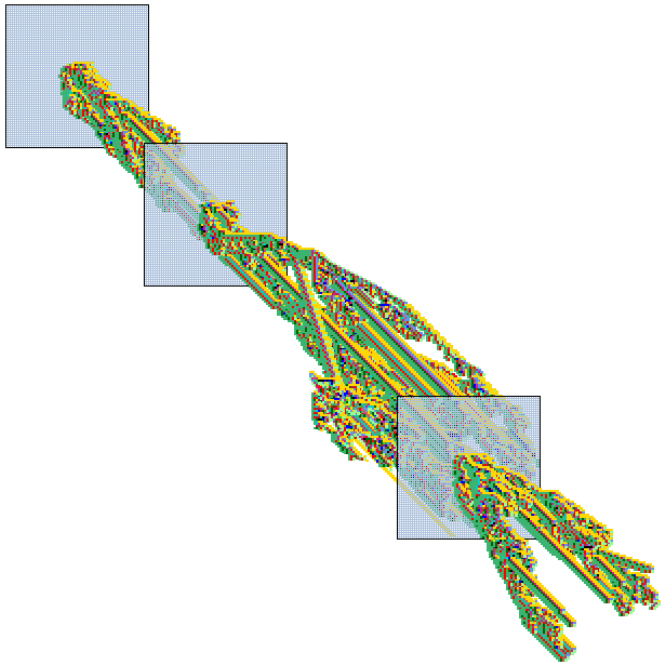


Fig. 13: Space-time patterns of the game-of-Life scrolling diagonally upwards (k=9, n=66×66). States are stacked in front of each other in an isometric projection, with new seeds set at intervals, and alternate time-steps skipped.

The prompts are fairly self-explanatory. They present themselves in a main sequence for the most common 1D CA parameters, and also in a number of context-dependent pop-up prompt windows for 2D, 3D, random wiring, various special settings, and during an output run if interrupted, or when the run stops. A flashing cursor prompts for input. To navigate the interface:

- enter **return** for a working default, which gives the next prompt. Repeat **return** to move forward through the prompt sequence. Eventually something interesting will happen even if no actual entries are made. All prompts have a default, some retain previous entries. An alternative to **return** is a click of the left mouse button.
- enter **q** to backtrack to the preceding prompt. Repeat **q** to backtrack up the prompt sequence. **q** will also revise an entry, and interrupt a running process being generated, such as space-time patterns or attractor basins. An alternative to **q** is a click of the right mouse button.
- any time a top-center notice is showing with **accept defaults-d**, enter **d** to skip a series of specialist prompts.
- enter appropriate input from the keyboard in response to a prompt, followed by **return** to move to the next prompt. Revise with **q** or **backspace**.
- To exit DDLab enter **Ctrl-q** at any prompt, followed by **q** (except in DOD – press **q** to backtrack until exit).

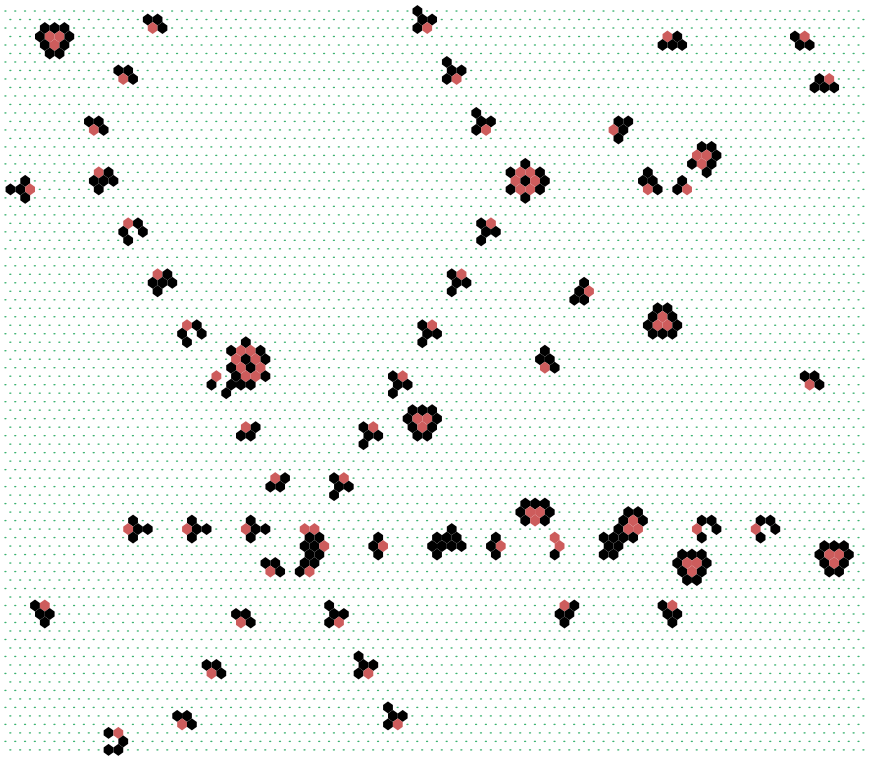


Fig. 14: A 2D CA space-time pattern on a hexagonal lattice. n=88×88, $[v, k]$=[3,7]. The k-totalistic rule 00020012002122022120022212202222121O gives rise to a zoo of interacting mobile and static emergent structures [27]: spiral glider-guns, mobile glider-guns, and self-reproduction by glider collisions.

2.1 TFO, SEED and FIELD modes

Some initial choices in the prompt sequence set the stage for all subsequent DDLab operations. TFO-mode constrains DDLab to run totalistic rules forward-only. This reduces memory load by cutting out full look-up tables and all attractor functions allowing larger neighborhoods as shown in table 2.

If DDLab is not constrained in TFO-mode, there is a further choice: FIELD-mode to show the whole basin of attraction field, or SEED-mode to show something that requires an initial state: a space-time pattern, a single basin of attraction, or a subtree. At the conclusion of an output run FIELD-mode and SEED-mode can be toggled without having to backtrack.

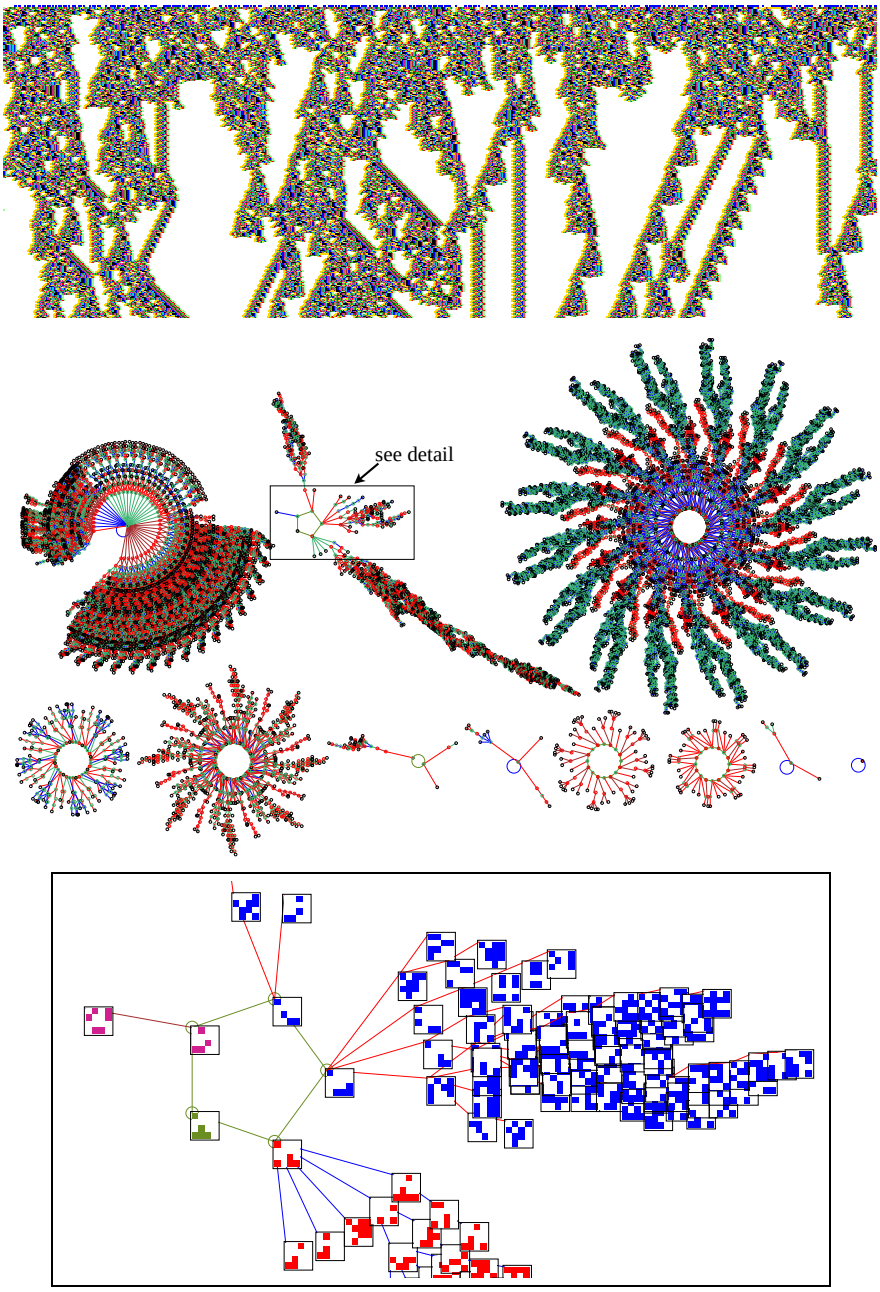


Fig. 15: *Top*: The space-time pattern of a 1D binary CA where interacting gliders emerge [23]. n=700, k=7, 250 times-steps from a random initial state. *Center*: The basin of attraction field for the same rule, n=16. The 2^{16} states in state-space are connected into 89 basins of attraction, but only the 11 nonequivalent basins are shown, with symmetries characteristic of CA [6.9sec]. *Bottom*: A detail of the second basin in the basin of attraction field, where states are shown as 4×4 bit patterns.

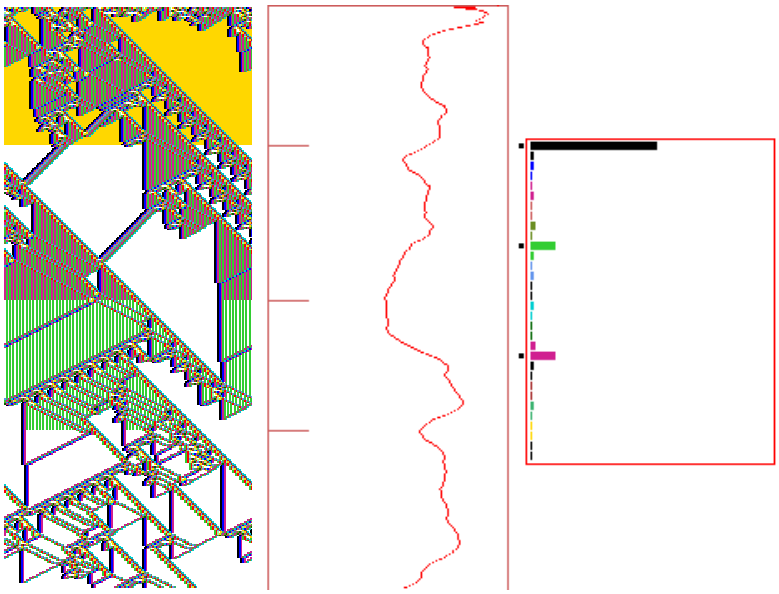


Fig. 16: A space-time pattern of a complex 1D CA, $[v,k]$=[2,5], n=150, hex rule e9 f6 a8 15, (as in Fig.9). 350 time-steps, and some analysis shown by default: *Left*: The space-time pattern colored according to neighborhood look-up, and progressively "filtered" on-the-fly at three times, suppressing the background domain to show up interacting gliders more clearly. *Center*: The input-entropy plot of – *Right*: the look-up frequency histogram, relative to a moving window of 10 time-steps. The filtered neighborhoods are indicated with a dot. The entropy variance is a sign of emergent structure.

3 Network Parameters

The parameters to define the network are presented in roughly the following sequence, but this depends on the mode chosen in section 2.1 – TFO, SEED and FIELD modes and subsequent choices. To make corrections backtrack with **q**. Depending on what is selected, some of these parameters may not apply, or there may be extra options in pop-up windows.

value-range v → system size n → neighborhood size k or k-mix
→ dimension 1D, 2D, 3D → wiring scheme
→ rule or rule-scheme → initial state

Wiring, rules and the initial state can also be amended during an output run or at its conclusion, including changes on-the-fly.

These parameters, described in the subsections below, may appear somewhat intimidating with their many alternatives and diversions, but DDLab provides defaults to all prompts and bypasses the more specialized prompts if not needed.

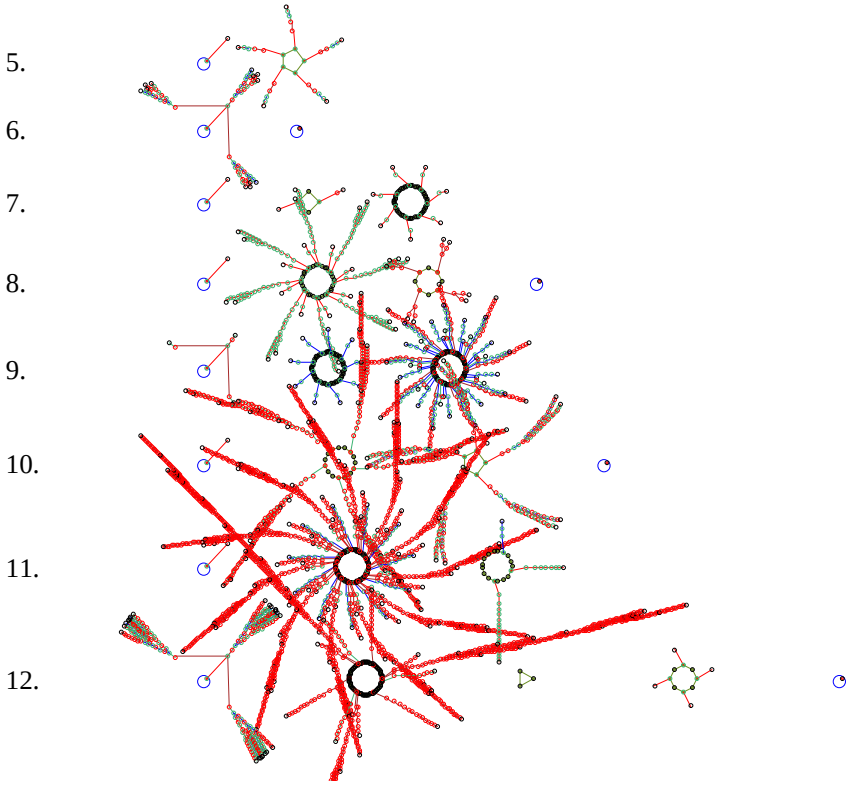


Fig. 17: Basin of attraction fields for a range of network size n=5-12, $[v, k]$=[2,3], rule 30 [0.59sec].

Fig. 18: 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.

3.1 Value-range, v

The value-range v is the number of possible cell-states, or colors, or letters in the cell's "alphabet." v can be set from 2 to 8 only at this early prompt. If v=2, DDLab behaves as in the old binary version. The value-range is indexed from 0 to 7, and is assigned a color code.

As v is increased, max-k will decrease (Table 1), and max-n in FIELD-mode will also decrease (Table 2).

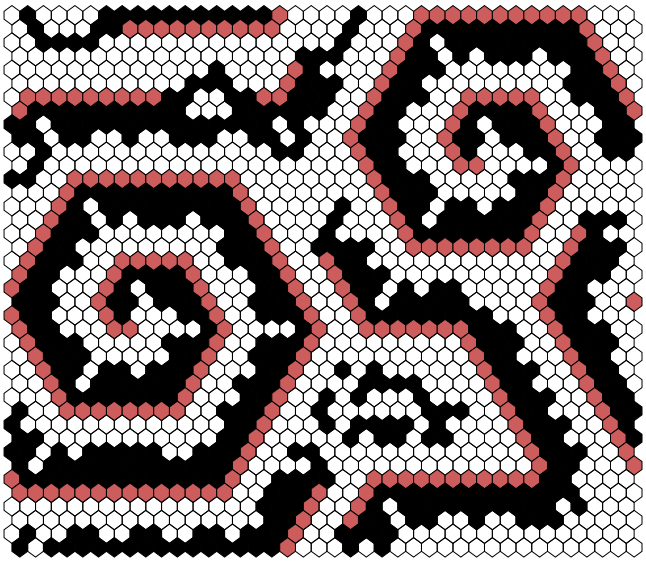


Fig. 19: Snapshot of a 2D CA on a hexagonal lattice. n=40×40, where spirals have emerged. $[v,k]$=[3,6]. The k-totalistic rule depends on just the frequency of the v=3 colors (2,1,0) in the k=6 neighborhood, as shown below:

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	} frequencies
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	
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	

3.2 Network Size, n

In FIELD-mode (for basin of attraction fields) the maximum network size (max-n) decreases with increasing value-range v as set out in Table 1. Note that in practice smaller sizes than max-n are recommended because of expected time and memory constraints.

In TFO-mode, and SEED-mode (for space-time patterns, single basins or subtrees), the network size is limited to max-n=65,025. A 2D square network is limited to (i,j)=255×255 and a 3D cubic network to (i,j,h)=40×40×40, but within the limits of n=65,025 there can be any combination of (i,j) and (i,j,h).

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

Table 1: For a basin of attraction field (FIELD-mode) the maximum network size, max-n, for different value-ranges v.

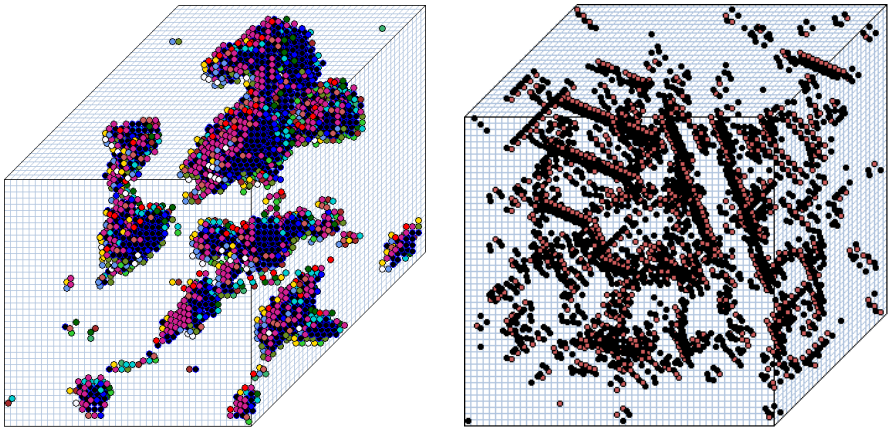
Max-n works fine for space-time patterns, but for single basins and subtrees, in practice much smaller sizes are appropriate, except when generating subtrees for maximally chaotic CA chain-rules (Sect. 8).

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.

unconstrained FIELD/SEED-mode		
v	max-k	table size S
2	13	8162
3	9	19683
4	7	16484
5	6	15629
6	5	16807
7	5	16807
8	4	4096

constrained TFO-mode		
v	max-k	table size S
2	25	26
3	25	351
4	25	3276
5	25	23551
6	17	26334
7	13	27132
8	11	31824

Table 2: The maximum neighborhood size, max-k, for value-ranges v. *Left*: for FIELD-mode or SEED-mode, with a full (unconstrained) rule-table. *Right*: for TFO-mode, for running totalistic rules forwards only, where the totalistic rule-tables are shorter, so max-k can be bigger. In both cases the size S of the rule-tables for v and max-k is shown.



$[v, k]$=[2,7] totalistic rule $[v, k]$=[3,6] totalistic rule

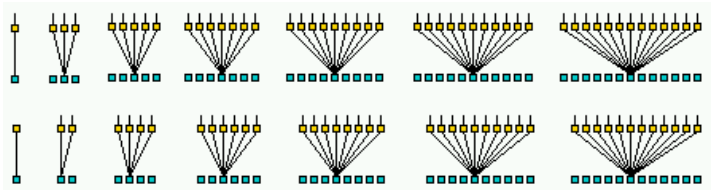
Fig. 20: Snapshots of 3D CA from random initial states. The projection is axonometric seen from below, as if looking up at the inside of a cage. *Left*: n=40×40×40. The binary totalistic rule 11101000, k=7, but with a bias of 1s of 45%. Cells are shown colored according to neighborhood look-up for a clearer picture, instead of by value: (0,1). *Right*: n=50×26×50, $[v, k]$=[3,6]. The k-totalistic rule 020000102010020000220012011O allows the emergence of gliders and other complex structures.

3.3 Neighborhood Size, k

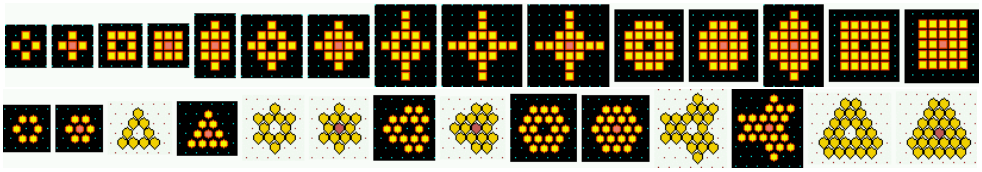
The neighborhood size, k, is the number of input wires to each cell. If k is not homogeneous the k-mix needs to be defined.

k can be set from 1 to max-k, but an effective k=0 is also possible. Permissible max-k decreases with increasing value-range v because look-up tables may become too large. A full look-up table v^k is larger than a k-totalistic look-up table $(v+k-1)!/(k!(v-1)!)$, so max-k depends on whether DDLab was constrained in TFO-mode, or unconstrained in SEED-mode or FIELD-mode. Max-k, for increasing value-range v, for both cases, is given in Tables 2, which also show the size of the corresponding rule-tables S.

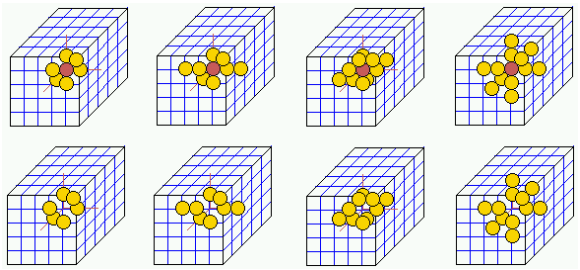
The k-mix may be set and modified in a variety of ways, including defining the proportions of different k's to be allocated at random in the network, or a scale-free distribution as in Fig. 4. A k-mix may be saved/loaded but is also implicit in the wiring scheme.



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. 21: Predefined 1D, 2D, and 3D neighborhood templates. For 1D and 2D, k-max$\leq$25 in TFO-mode, otherwise k-max$\leq$13. For 3D k-max$\leq$13. For 2D the neighborhood/lattice can be either square or hexagonal.

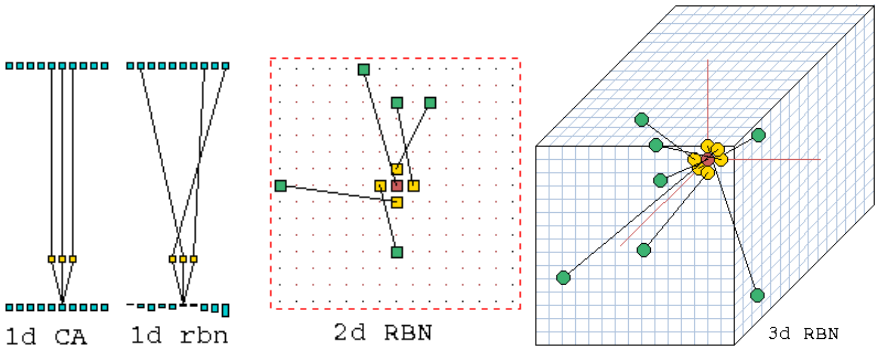


Fig. 22: The wiring of a cell: for random wiring, cells anywhere in the network are wired back to each position in a "pseudo-neighborhood," which corresponds to the CA neighborhood. Each cell's wiring can be shown (and amended) by moving around the network with the mouse or keyboard. *Left:* 1D: The wiring is shown between two time-steps. *Center:* 2D: k=5. *Right:* 3D: k=7.

3.4 Dimensions

Fig. 21 shows some of the predefined neighborhoods templates in 1D, 2D, or 3D, where each neighbor is indexed from 0 to k-1. The templates, which are designed to maximize symmetry, are assigned automatically when the dimensions are selected (the default is 1D), and conform to k or the k-mix. In 2D there is a choice of square or a hexagonal neighborhood template. A hexagonal template results in a lattice with hexagonal tiling (Figs. 19 and 14). The choice of dimensions also sets the network's *native* geometry, with periodic boundary conditions, where lattice edges wrap around to their opposite edges. The *presentation* geometry can be different from the native geometry. The neighborhoods templates also serve as the pseudo-neighborhoods for random wiring, because the rule requires an indexed neighborhood.

3.5 Wiring or Connections

The wiring scheme, the networks connections, defines the location of the output terminals of each cell's input wires (Fig. 22), which connect to each neighbor in the indexed neighborhood or pseudo-neighborhood. For CA, the output terminals correspond to the neighborhood, but for random wiring they could be anywhere within the network.

Wiring is perhaps the least studied aspect of the dynamics *on* networks. DDLab makes it possible to weave the tapestry.

1D, 2D or 3D networks, whether CA or randomly wired, can be set up automatically, with a wide variety of constraints and biases. For a example, random wiring can be confined within a local patch of cells with a set diameter in 1D, 2D or 3D (Fig. 24 *Right*). Some wires can be released from the

neighborhood or patch. 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.

A wiring scheme can be set and amended just for a predefined sub-network within the network, which may be saved/loaded, and hybrid networks of CA/DDN can be created, or a network of sub-networks in any combination.

Random wiring can be set or amended by hand. The wiring may be hypercube if $k = log_2 n$ or $log_2 n + 1$, for example k=3, n=8 or 9.

The network parameters including wiring can be displayed and amended in a 1D, 2D or 3D graphic format (Fig. 22 and 26) or in a "spreadsheet" (Fig. 27). The wiring can also be shown in a network-graph which can be rearranged in many ways, including dragging nodes with the mouse as in Figs. 1, 2, 4 and 28.

3.6 Rule or Rule-mix

A network may have one rule or a rule-mix. The rule mix can be confined to a subset of preselected rules. Rules may be set and modified in a wide variety of ways, in decimal, hex, as a rule-table bit or value string, 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.

In its most general form of update logic, a rule is expressed as a full look-up table, but there are subsets of the general case, two types of totalistic rules, and their corresponding outer-totalistic rules.

The simplest, t-totalistic rules, depend on the sum of values in the neighborhood. k-totalistic rules depend on the frequency of each value (color) in the neighborhood (see Fig. 19). If k=2 these two types are identical.

Both types of totalistic rules can be made into outer-totalistic rules, where a different rule applies for each value of the central cell; the Game-of-Life is one such rule (Fig. 13).

Totalistic and outer-totalistic rules can be run in either SEED-mode or TFO-mode – transformations and mutations then apply to to either the full or the totalistic look-up table.

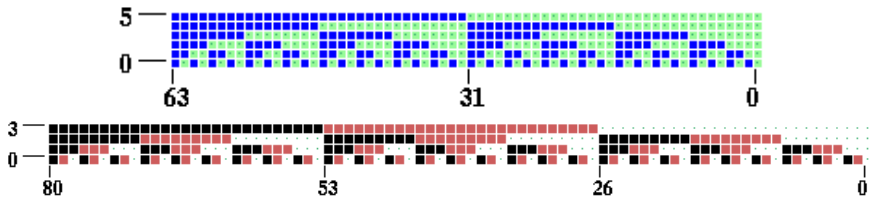


Fig. 23: Examples of the neighborhood matrix of full look-up tables (rule-tables), showing all possible neighborhoods (vertically). The position of each neighbor is indexed from k-1 to 0. *Above:* $[v, k]$=[2,6] all 64 neighborhoods, from all-1s to all-0s. *Below:* $[v, k]$=[4,3] all 81 neighborhoods, from all-3s to all-0s.

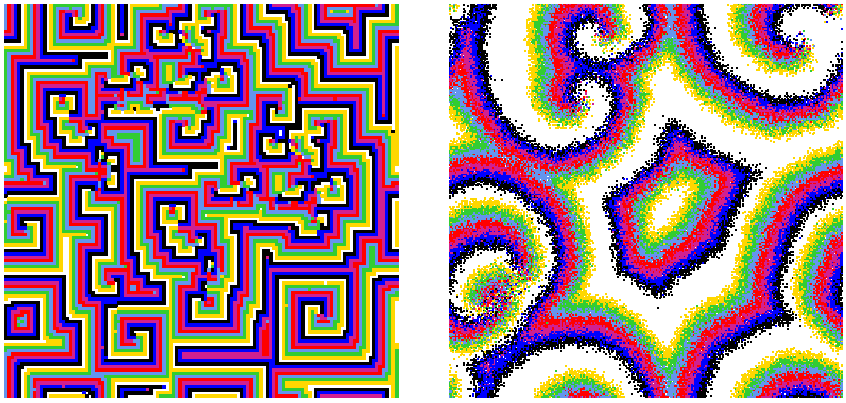


Fig. 24: Reaction-Diffusion or excitable media [3] dynamics. *Left*: [v,k]=[8,8], threshold interval is 1 to 6, n=122x122. *Right*: [v,k]=[8,11], threshold interval is 2 to 7, n=255x255, random connections within a 24 diameter local zone.

There are also subsets of rules that can be automatically selected at random, including isometric rules – where rotated and reflected nhoods (1D, 2D or 3D) have the same output, chain-rules [28] described in Sect. 8, and Altenberg rules (Fig. 35).

Rules can be biased by various parameters, λ [9], Z [16, 23], canalizing inputs [4], and Post functions [12]. The Game-of-Life, majority (Fig. 25), and other predefined rules can be selected.

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 (Fig. 33). Rule parameters λ and Z, and the frequency of canalizing inputs in a network, can be set to any arbitrary level.

DDlab is also able to implement reaction-diffusion or excitable media dynamics [3] (Fig. 24), which can be set as an outer-k-totalistic rule in TFO-mode, or as a full lookup-table in SEED-mode.

3.7 Initial 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 subtree, or a single basin by first running forward to find the attractor, then backward to generate the subtree from each attractor state. A basin of attraction field does not require setting a seed, because appropriate seeds are automatically provided.

To generate a subtree from a random seed it is usually necessary to run forward by a few steps to penetrate the subtree before running backward (Fig. 12). This is because for most DDN and CA (except chain-rules [28]

Fig. 25: *Top Left*: A 2D seed, n=88×88, drawn with the mouse and keyboard, $[v, k]$=[8,4]. Colors 0 to $(v-1)$ can be selected. 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 the seed is then transformed by applying a CA majority rule for three time-steps.

Sect. 8), most states in state space have no predecessors; they are the subtree leaves or garden-of-Eden states (Fig. 40).

As in setting a rule, there are a wide variety of methods for defining the seed: in decimal or hex, drawing bits or values in 1D, 2D, or 3D – a mini-paint program (Fig. 25), at random (with various constraints or biases), or loaded from a file.

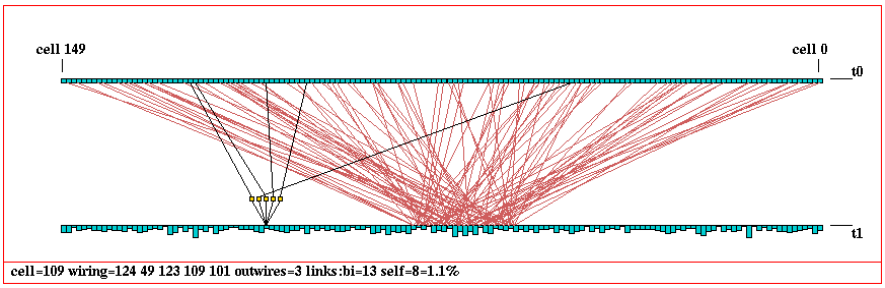


Fig. 26: The 1D wiring graphic, showing wiring to a block within a 1D network. k=5, n = 150. The block was defined from cells 60–80. Revisions to rules and wiring can be the confined just to the block. The "active cell" is still visible and can be moved as usual. The 1D wiring graphic can also be shown as a circle.

4 Network Architecture

Once the network is defined, DDLab provides tabular and graphical methods for examining, amending, duplicating, unraveling and displaying the network. These functions can be accessed from the initial sequence of prompts, or during an output run if interrupted or when the run stops.

4.1 Wiring Graphic

The wiring graphic (Fig. 22 and 26) can be displayed in 1D, 2D, or 3D. The network's wiring and rules can be examined, changed, and tailored to requirements, including biased random settings to predefined parts of the network. Individual cells can be examined by moving around the diagram with the keyboard, and the cell's rule or individual wires modified. A block of cells can be defined and its wiring randomized or relocalized.

These are very flexible methods, and for RBN/DDN it is usually easier to set up a suitable dummy network initially, then tailor it in the wiring graphic.

From the wiring graphic it is possible to create a system of independent or weakly coupled sub-networks 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 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 k-mix sub-network into a homogeneous-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.

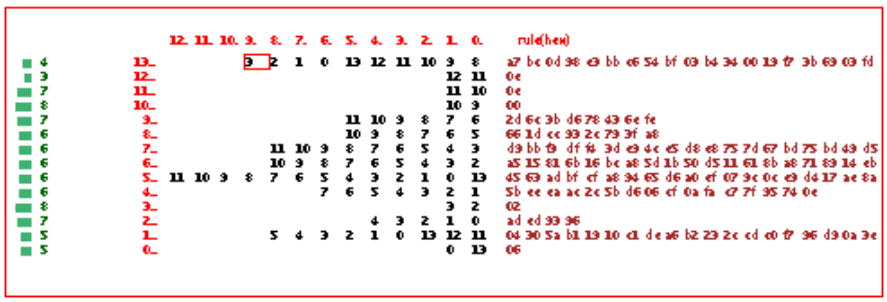


Fig. 27: The wiring matrix for a mixed k network with random wiring. n=14, k=2–13. Rows 13...0 show the wiring and rule for each cell. Columns 12...0 show from where each pseudo-neighborhood position is wired. The far left column shows the "out-degree" of each cell, the number of output wires that link to it, as a histogram. The far right column shows the rules in hex (as much as will fit). It is possible to move around the wiring matrix as in a spreadsheet to change wiring settings.

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.

Connectivity measures from the wiring graphic include the following:

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

4.2 Wiring Matrix

The wiring matrix (Fig. 27) displays the wiring and rule information in a tabular format, like a spread-sheet. For each cell tabulated against its pseudo-neighborhood index, the position of the input cell is shown. You can move around the matrix with the keyboard and amend this position.

4.3 Network-Graph

Another method of reviewing network architecture is an adjacency matrix and network-graph (Figs. 1, 2, 4, and 28) 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.

The methods and options are similar to those of the jump-graph (section 5.3) – in fact the functions below apply equally to the jump-graph.

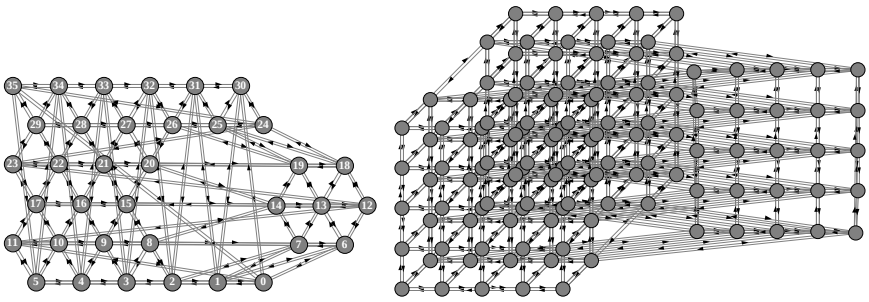


Fig. 28: Network-graphs of a 2D and 3D CA. *Left*: a 2D hexagonal network, showing a node and its 1-step inputs dragged out. *Right*: a 3D network shown as an axonometric projection seen from below as if looking up into a cage. A vertical slice has been defined and dragged from the graph.

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 link distance of a fragment 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, jiggled, and random. 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.

5 Space-Time Patterns and Attractor Basins

Once the network is defined, a further series of prompts set the presentation and analysis in advance of the final output. The prompts depend on the context and previous selections and are presented in a succession of pop-up windows. They are arranged into the following categories.

- For attractor basins: various, layout, display, pause/data and mutation.
- For space-time patterns: various, analysis, and histograms.

These options all have default setting, so you can skip them entirely or or skip just the remaining categories at any point with **d**, or backtrack with **q** and skip to the required category.

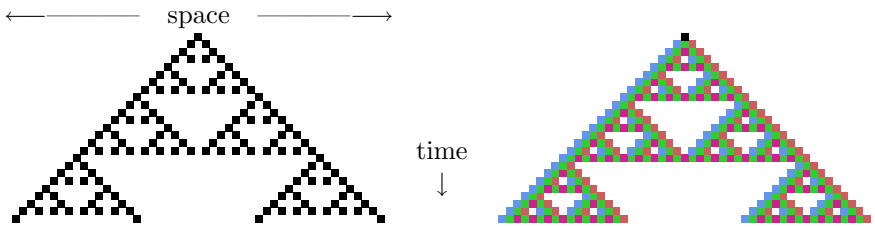


Fig. 29: Space-time patterns of a 1D binary CA, $[v, k]$=[2,3], n=51, rule 90. 24 time-steps from an initial state with a single central 1. Two alternative presentations are shown. *Left*: cells by value. *Right*: cells colored according to their look-up neighborhood.

After these prompts, space-time patterns start. Attractor basins start after some further final options. During the running process there are many interrupt and on-the-fly options, especially for space-time patterns. These are too numerous to list in full, but the subsections below provide some flavor.

5.1 Options for Space-Time Patterns

There are many alternative forms of presentation of space-time patterns and these can be changed on-the-fly, with a key press, while the space-time patterns are running. 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 (Fig. 29). Space-time patterns can be filtered to suppress cells that updated according to the most frequently occurring neighborhoods, thus exposing "gliders" and other structures (Figs. 16 and 30).

The presentation can 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 are a sign of order, which can be induced by "canalizing inputs," applied in Kauffman's RBN model of gene regulatory networks [6, 4]. A 1D space-time pattern is presented scrolling vertically, or by successive vertical sweeps. 2D networks can be toggled between square and hexagonal layout, and presented with a time dimension (2D+time), scrolling either vertically or diagonally (Fig. 13). 3D networks are presented within a 3D "cage" (Fig. 20), but can also be shown in slices. 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.

Space-time patters may iterate too fast for comfort, especially for 1D. They can be slowed down progressively on-the-fly or restored to full speed with the arrow keys.

There are many other on-the-fly options which do not interrupt the running space-time patterns, in whatever form of presentation. These include: skipping time-steps, pausing after x time-steps, reversing to previous time-

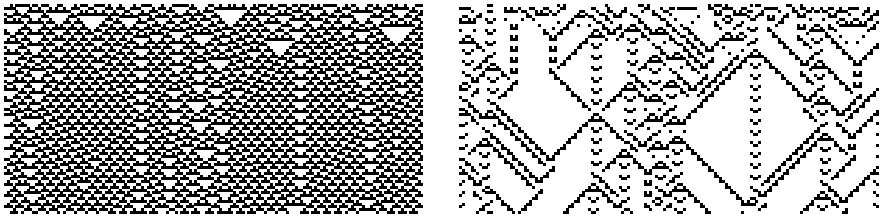


Fig. 30: Filtering a binary 1D space-time pattern with interacting gliders embedded in a complicated background. $Left$: unfiltered. *Right*: the same space-time pattern filtered. Filtering is done on-the-fly for any rule. In this example, k=3 rule 54 was first transformed to its equivalent k=5 rule (hex: 0f3c0f3c). n=150, 75 time-steps from a random initial state.

steps, changing the scale, changing or perturbing the current state, changing rules, toggling wiring between local and random, and showing cell stability. Bits/values in a rule can be progressively flipped and unflipped. Various qualities of noise can be added. Synchronous updating can be toggled with asynchronous in a random or a predefined partial-order.

Concurrently with the standard presentations, space-time patterns can be displayed in a separate window according to the network-graph layout, which 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, then scrolled diagonally (Fig. 9).

5.2 Options for Attractor Basins

Many options are provided for the layout of attractor basins, their size, position, spacing, rotation, and type of node display – as a spot, in decimal, hex, or a 1D or 2D bit pattern. Just the transient nodes, garden-of-Eden nodes (or none) can be displayed. The in-degree spread or fan-angle can be adjusted.

Subtrees, single basins and the basin of attraction field can be draw for a range of network sized (Fig. 17). The basin of attraction field can be generated within the movable nodes of a jump-graph or layout-graph to achieve any arbitrary layout (Figs. 31 and 32).

Regular 1D and 2D CA produce attractor basins where subtrees and basins are equivalent by rotational symmetry [16]. This allows "compression" of basins into just nonequivalent prototypes, and also their subtrees. 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 these data, including the complete list of states in basins and subtrees, can be displayed in the terminal window or saved to a file. When pausing, the default position,

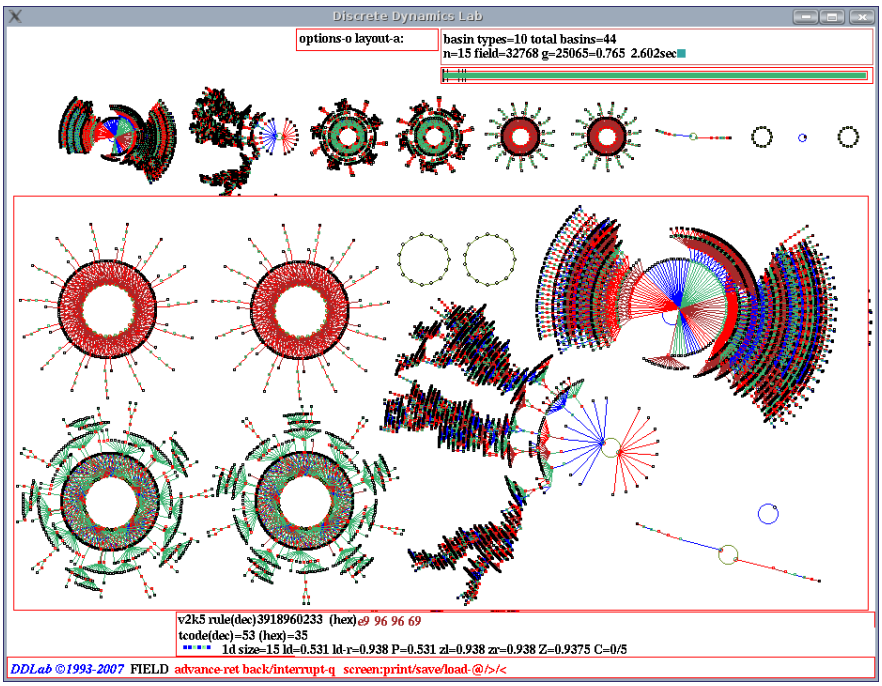


Fig. 31: The DDLab screen showing two layouts of a basin of attraction field. *Top*: the normal layout but with spacing adjusted after each basin. *Inset below*: the layout-graph where nodes (thus basins) can be dragged into any position. Both results can be saved as vector PostScript files. This example is for a binary 1D CA, k=5, totalistic rule 53, n=15 [2.4 sec].

spacing, rotation and other properties of the next basin can be amended.

When an attractor basin has finished, a key-press will generate a "mutant" according to the mutant setting – for example flipping a bit or value in the rule table, or moving a wire. The pause may be turned off to create a continuous demo. A "screensave" option shows mutant basins continually growing at random positions.

Attractor basins can be generated according to synchronous updating, or asynchronous updating in a predefined partial-order.

5.3 Attractor Jump-Graph and Layout-Graph

The stability of basins of attraction is especially relevant to the stability of cell types in biology, following the genetic regulatory network approach [5, 11].

DDLab provides 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,

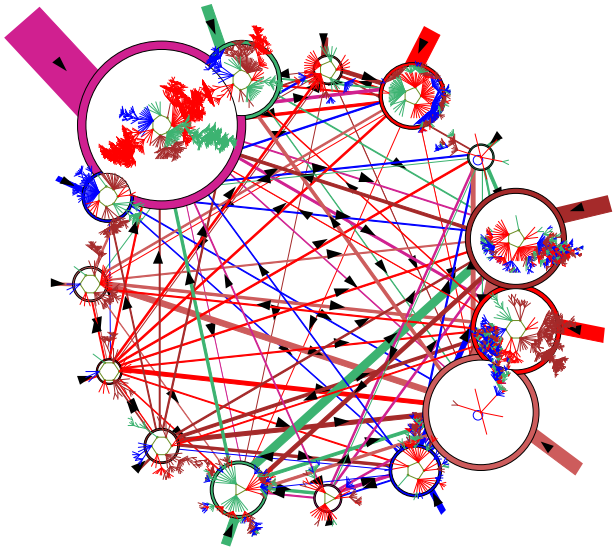


Fig. 32: The jump-graph with the basins of attraction field (RBN k=3, n=13, in Fig. 5) redrawn within its nodes. The jump-graph shows the probability of jumping between basins due to single bit-flips to attractor states. Nodes representing basins are scaled according the number of states in the basin (basin volume), and can be rearranged and dragged. Links are scaled according to both basin volume and the jump probability. Arrows indicate the direction of jumps. Short stubs are self-jumps. When jump-graph links are eliminated this becomes a layout-graph with mobile nodes where basins can be redrawn at node positions.

and graphically as the attractor jump-graph: a graph with weighed vertices and edges giving a graphic representation of the jump-table (Fig. 32).

The jump-graph can be analyzed and manipulated in many 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, Sect. 4.3.

Basins of attraction can be redrawn within the jump-graph nodes (Fig. 32). If the jump links are eliminated this becomes a layout-graph (Fig 31) where dragging/rearranging nodes provides an alternative flexible method for positioning basins – the resulting graphics can be saved as vector PostScript, for example Figs. 7, 10 and 15-*Center*.

5.4 Mutation

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)

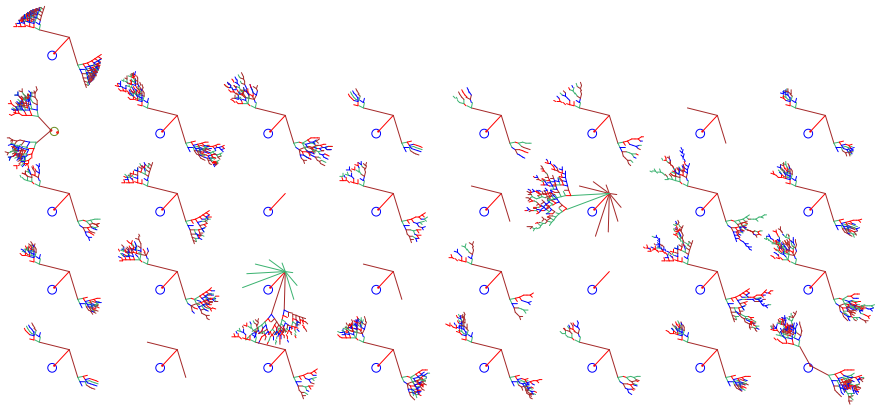


Fig. 33: Mutant basins of attraction of the $[v, k]$=[2,3] rule 60 (n=8, seed all 0s). *Top left*: The original rule, where all states fall into just one very regular basin. The rule was first transformed to its equivalent k=5 rule (f00ff00f in hex), with 32 bits in its rule table. All 32 one-bit mutant basins are shown. If the rule is the genotype, the basin of attraction can be seen as the phenotype [0.02sec]

are provided as files with DDLab, and these can be activated on-the-fly.

When attractor basins are complete, a key-press will regenerate the attractor basin of a mutant network. Various mutation options can be preset including random bit/value-flips in rules and random rewiring of a given number of wires. Sets of states can be specified and highlighted in attractor basins to see how mutations affect their distribution. The complete set of one-bit mutants of a rule can be displayed together as in Fig. 33.

5.5 Learning and Forgetting

Learning and forgetting algorithms allow attaching and detaching sets of states as predecessors of a given state by automatically mutating rules or altering wiring couplings [18, 20]. This allows "sculpting" the basin of attraction field to approach a desired scheme of hierarchical categorization, the network's content addressable memory. 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 are useful for fine tuning a network that is already close to where it is 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 pruning the connections in a fully connected network to satisfy an exhaustive map (for network sizes $n \leq 13$). The inverse problem is to find a minimal network that will satisfy a full or partial mapping, fragments of attractor basins or trajectories.

5.6 Sequential Updating

By default, network updating is synchronous, in parallel. DDLab also allows sequential updating, both for space-time patterns and attractor basins. Sequential orders are forward, backward, or a random order, but any specific order can be set from the $n!$ possible sequential 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].

DDLab can also set a sequential partial-order, where cells are allocated to groups which update sequentially in a give order, but with synchronous updating within each group. This allows the updating to be tuned between fully synchronous and fully sequential.

Setting an update probability can also achieve asynchronicity, where only part of the network, randomly assigned, updates at each time-step.

5.7 Vector PostScript of Graphic Output

Graphic output in DDLab can be captured as vector PostScript, which is preferable for publication quality images over bitmap graphics, as they can be scaled without degrading the resolution. Many of the figures in this chapter are vector images, others are bitmap graphics captured with a screen grabber. At the time of writing, vector PostScript files are available for the following: space-time patterns in 1D and 2D, attractor basins of all kinds, the network-graph, attractor jump-graph, and layout-graph.

Vector PostScript files are ascii files in the PostScript language, so can be edited for changes and additions. If selected, functions in DDLab generate these files automatically according to the the current presentation of the DDLab graphic, but also with various options for the PostScript image itself such as color or grayscale. There is no limit on the size of a PostScript file, so for large attractor basins with many links and nodes, proceed with caution to avoid excessively large files. If an attractor basin is interrupted, the PostScript file of part of the graphic generated up to that point will be valid.

5.8 Filing Network Parameters, States and Data

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.

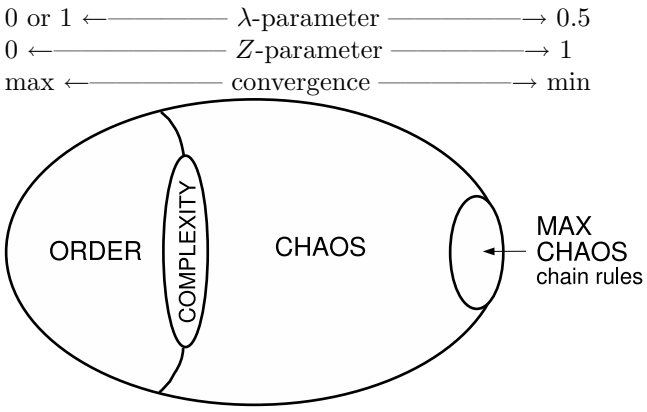


Fig. 34: A view of rule-space after Langton and his λ-parameter [9]. Tuning the Z-parameter from 0 to 1 moves the dynamics from maximum to minimum convergence, from order to chaos, traversing a phase transition where complexity might be found. The chain-rules, added on the right, are maximally chaotic and have the very least convergence, decreasing with system size, making them suitable for dynamical encryption.

Various data including mean entropy and entropy variance of space-time patterns can be automatically generated and saved, allowing a sorted sample of CA rules to be created, discriminating between order, complexity, and chaos [23], as in Fig. 37. 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.

6 Measures and Data

DDLab provides static measures for rules, wiring and various quantitative, statistical and analytical measures and data on both space-time patterns and attractor basin topology. The measures and data are shown graphically in most cases as well as numerically.

Sect. 4 listed connectivity measures from the wiring graphic. Other measures are listed in the following subsections.

6.1 Static Rule Measures

DDLab provides various static measures depending just on the rule-table, or rule-tables in mixed-rule networks. These measures or parameters are generalized for multi-value.

- The λ-parameter [9] which measures the density of non-quiescent values in the rule-table. λ approximates the Z-parameter.
- The λ-ratio [16] equivalent to λ, but defined differently.

- The P-parameter [4] equivalent to λ for binary systems, but defined differently again.
- The Z-parameter [16], a probability measure, which predicts the convergence, bushiness, of sub-trees in attractor basins, thus order/chaos in the dynamics (Fig. 34).
- the (weighted) average λ and Z in a rule-mix.
- canalizing inputs [6, 4] and the frequency of canalizing "genes" and inputs, which can be tuned to required values and displayed graphically. Canalizing inputs induce order in RBN/DDN (Fig. 38).
- Post functions [12], which also induce order in RBN, and where the dynamics are closed under composition.

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

6.2 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 input-entropy plot (Fig. 16). This is the basis of the method for automatically filtering space-time patterns [23] (Fig. 16 and 30).

- The space-time color density in a moving window of time-steps(Fig. 35).

- The variance or standard deviation of the entropy, and an entropy/density scatter plot [23], where complex rules have their own distinctive signatures (Fig. 36).

- A scatter plot of mean entropy against the standard deviation (or an alternative "min-max" measure) of the entropy for an arbitrarily large sample of CA rules, which allows ordered, complex, and chaotic rules to be classified automatically [21, 23], also shown as a 2D frequency histogram (Fig. 37). Ordered, complex and chaotic dynamics are located in different regions, allowing a statistical measure of their frequency. The rules can be sorted by the various measures, allowing complex rules to be found automatically.

- Various methods for showing the activity/stability of network cells, or frozen "genes" [4].

- The damage-spread [21], 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.

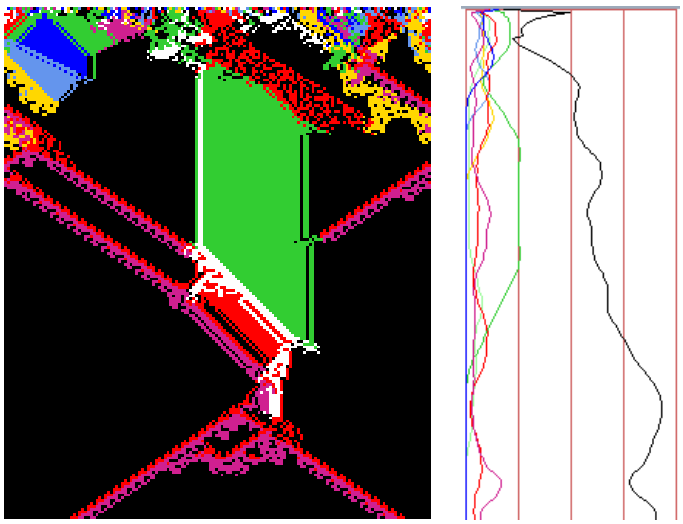


Fig. 35: *Left*: A 1D CA space-time pattern, from a random initial state, of an Altenberg k-totalistic rule $[v, k]$=[8,7], n=150, where the probability of a rule-table output depends on the fraction of colors in its neighborhood, resulting inevitably in emergent structure. *Right*: The color density plotted for each of the 8 colors, relative to a moving window of 10 time-steps.

- The Derrida plot [4, 6], 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, indicating if a RBN/DDN is in the ordered or chaotic regime (see Fig. 38).

- A scatter plot of successive iterations in a 2D phase plane, the "return map" (Fig. 39), which has a fractal structure for chaotic rules.

6.3 Measures on Attractor Basins

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

- 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

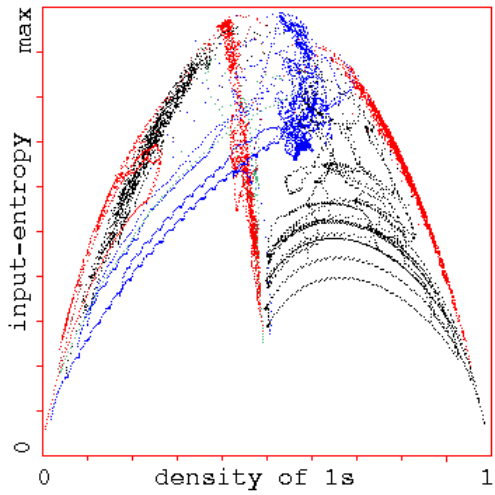


Fig. 36: Entropy/density scatter plot [23]. Input-entropy is plotted against the density of 1s relative to a moving window of 10 time-steps. Plots for a number of $[v, k]$=[2,5] complex rules (n=150) are show superimposed in different colors, each of which has its own distinctive signature, with a marked vertical extent, i.e. high entropy variance. About 1000 time-steps are plotted from several random initial states for each rule.

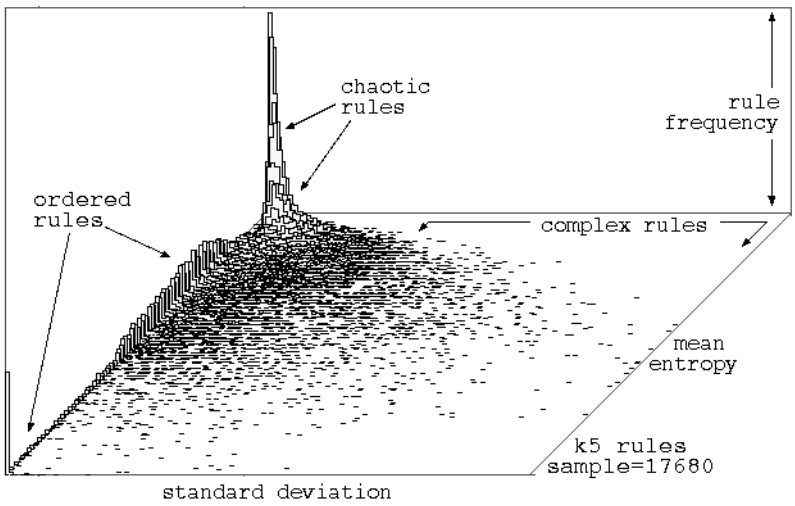


Fig. 37: Classifying a random sample of $[v, k]$=[2,5] rules automatically by plotting mean entropy against the standard deviation of the entropy. The 2D histogram shows the frequency of rules falling within a 128×128 grid. The rules can be sorted, and their space-time patterns selectively examined.

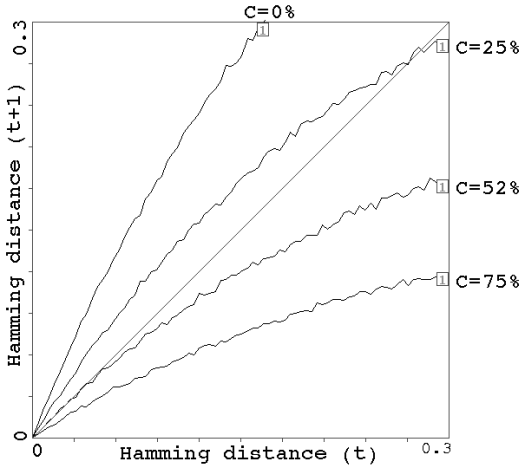


Fig. 38: Derrida plots for random Boolean networks ($[v, k]$-[2,5], n=36×36). This is a statistical measure of how pairs of network trajectories diverge/converge in terms of their Hamming distance. A curve above the main diagonal indicates divergence and chaos, below the diagonal – convergence and order. A curve tangential to the 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, with the 52% curve balanced at the edge of chaos.

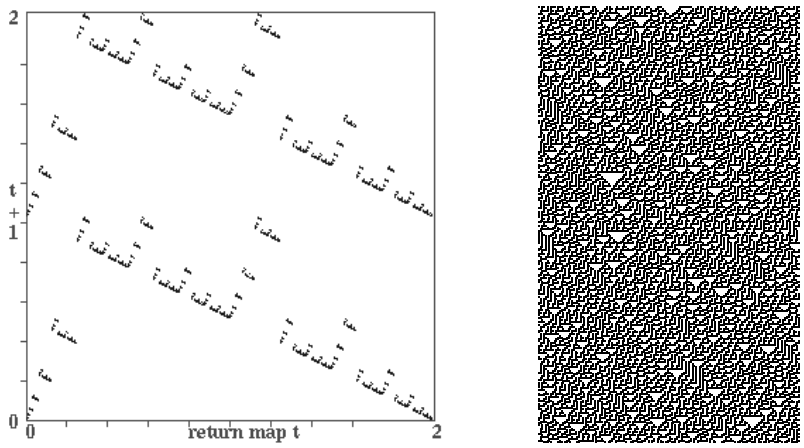


Fig. 39: *Left*: The return map for the space-time patterns of the binary 1D k=3 rule 30, which is a maximally chaotic chain-rule, n=150. The space-time pattern was run for about 10,000 time-steps. Note the fractal structure. Each state (bit-string) $B_0, B_1, B_2, B_3, \ldots B_{n-1}$ is converted into 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 time-step t+1 (y-axis). *Right*: the space-time pattern itself, showing 250 time-steps.

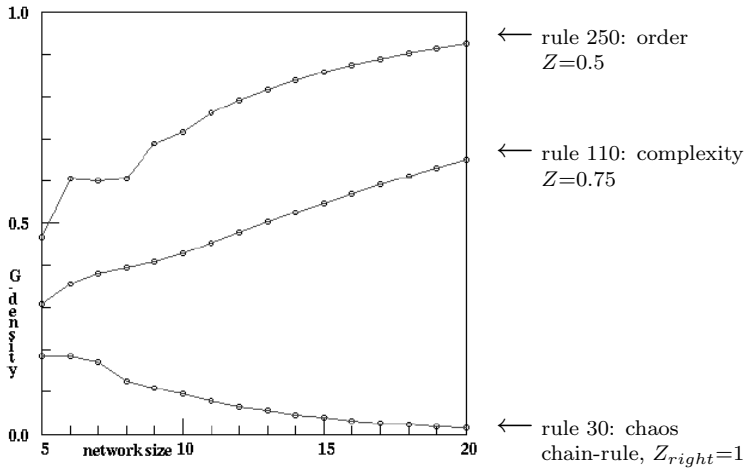


Fig. 40: A plot of leaf (garden-of-Eden) density with increasing system size, for an ordered, complex and maximally chaotic chain-rule, n=5 to 20. The measures are for the basin of attraction field, so for the entire state-space. For rule-space in general, leaf-density increases with greater n, but at a slower rate for increasing Z, but uniquely, leaf-density decreases for chain-rules (Sect. 8) where either Z_{left}=1 or Z_{left}=1, but not both.

attraction field for large networks. The number of basins, their relative size, period, and the average run-in length are measured statistically. The data can be used to automatically generate an attractor jump-graph as in Fig. 32. An analogous method shows the frequency of arriving at different "skeletons," consisting of patterns that have frozen to a specified degree.

- Garden-of-Eden or leaf-density plotted against the λ and Z parameters, and against network size as in Fig 40.

- 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 (Fig. 32) as described in section 5.3.

7 Reverse Algorithms

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

- An algorithm for 1D CA, or networks with 1D CA wiring but mixed rules.
- A general algorithm for RBN/DDN, which also works for the above.
- An exhaustive, brute force, 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 1D CA algorithm [16, 23] is the most efficient and fastest. Furthermore, compression of 1D CA attractor basins by rotation symmetry speeds up the process [16].

Any other network architecture, RBN or DDN, with non-local wiring, will be handled by the slower *general* reverse algorithm [17, 23]. A histogram revealing the inner workings of this algorithm can be displayed (Fig. 41). 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.

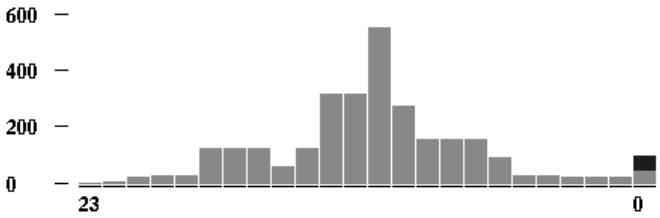


Fig. 41: Computing RBN pre-images. The changing size of a typical partial pre-image stack at successive elements. n=24, k=3. This histogram can be automatically generated for a look at the inner workings of the RBN/DDN reverse algorithm.

The 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/loaded). 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 and partial order 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.

8 Chain-Rules and Dynamical 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 $[v, k]$. These are rules based on special values of the Z-parameter, where either Z_{left}=1 or Z_{right}=1, but not both [16, 28]. The approximate number of chain-rules for binary CA is the square root of rule-space $2^{2^{k-1}}$.

Chain-rules are special because in contrast to the vast majority of rule-space (Fig. 34), 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 leaf-density is also very low, also decreasing with system size (Fig. 40), becoming vanishingly small in the limit. This means nearly all states have predecessors, embedded deeply along chain-like transients.

Large 1D CA can be therefore run backward very efficiently with chain-rules, generating a chain of predecessors. As the rules rapidly scramble patterns, they allow a method of encryption [28] which is available in DDLab; run backward to encrypt, forward to decrypt (Figs. 42 and 43). When decrypting, the information pops out suddenly from chaos and merges back into chaos. A chain-rule (of which there is a virtually inexhaustible supply) is the cypher key, and to a lesser extent the number of backward steps.

9 DDLab Website, Manual, and Reviews

DDlab continues to be developed with updates at irregular intervals. A new manual for the multi-value version is in the process of being written, and parts

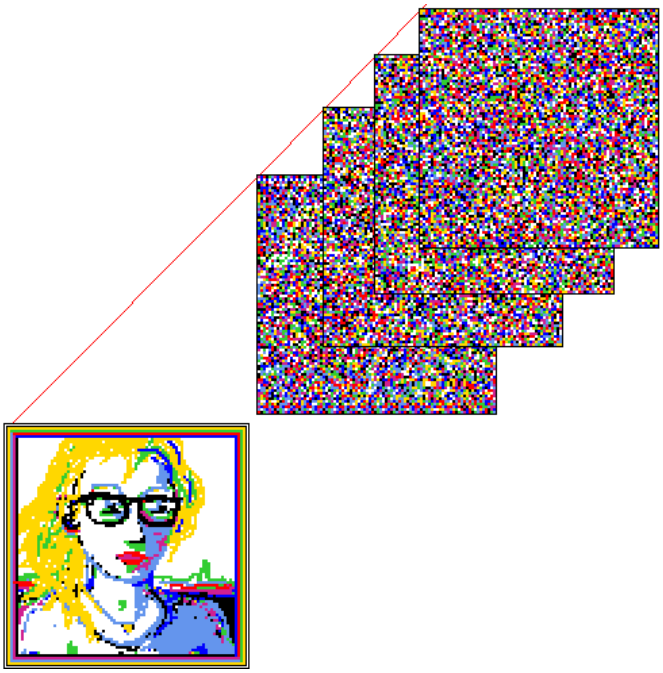


Fig. 42: The portrait was made with the drawing function in DDLab (as Fig. 25). However, this is a 1D pattern displayed in 2D, n=7744, 88×88. As v=8 there are 3 bits per cell, so the size of the bit-string=23,232, which could be an ASCII file, or any other form of information. With a $[v, k]$=[8,4] chain-rule constructed at random, and the portrait as the root state, a subtree was generated with the CA reverse algorithm, and set to stop after 4 backward time-steps. The subtree has no branching, and branching is highly unlikely to occur. The state reached is the encryption [50.5sec].

Fig. 43: To decrypt, starting from the encrypted state in Fig. 42, the CA with the same chain-rule was run forward to recover the original image. This figure shows time steps -2 to +7, to illustrate how the portrait was scrambled both before and after time-step 4

are already posted on DDLab's website. This, together with the existing manual will provide a useful guide.

The latest information, downloads, manuals, updates, examples, lecture slides, galleries, reviews, publications, and preprints, can be found in [25].

Some reviews of DDLab and "The Global Dynamics of Cellular Automata" [16]:

- John E. Myers in Complexity, Vol.3, No.1, Sept/Oct 1997.
- Andrew Adamatzky in Kybernetes, Vol.28, No.8 and 9, 1999.
- Stuart Kauffman in Complexity Vol.5, No.6, July/Aug 2000.
- H. Van Dyke Parunak in JASSS, The Journal of Artificial Societies and Social Simulation, Vol.4, Issue 4, Oct 2001.

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

This chapter is dedicated to the memory of
Brosl Hasslacher

References

[1] Adamatzky A (1994) Identification of Cellular Automata. Taylor and Francis, London.

[2] Burraston D and Martin A (2006) Digital Behaviors and Generative Music, Wild Nature and the Digital Life, Special Issue, Leonardo Electronic Almanac Vol 14, No. 7–8, `http://leoalmanac.org/journal/Vol_14/lea_v14_n07-08/dburastonamartin.asp`

[3] Greenberg JM and Hastings SP (1978) Spatial patterns for discrete models of diffusion in excitable media, SIAM J. Appl. Math. Vol 34, No 3, 515–523.

[4] Harris SE, Sawhill BK, Wuensche A, and Kauffman SA, (2002) A Model of Transcriptional Regulatory Networks Based on Biases in the Observated Regulation Rules, Complexity, Vol.7/no.4, 23–40.

[5] Kauffman SA (1969) Metabolic Stability and Epigenesis in Randomly Constructed Genetic Nets, Theoretical Biology, 22(3), 1969, 439–467.

[6] Kauffman SA (1993) The Origins of Order. Oxford University Press.

[7] Kauffman SA (2000) Book and Software Reviews, The global dynamics of cellular automata, by Andrew Wuensche and Mike Lesser, Complexity Vol.5, No.6, 47–48. `http://www.cogs.susx.ac.uk/users/andywu/kauffman_rev.html`

[8] Langton CG (1986) Studying Artificial Life with Cellular Automata, Physica D 22, 120–149.

[9] Langton CG (1990) Computation at the edge of chaos: phase transitions and emergent computation, Physica D 42, 12–37.

[10] Langton CG (1992) Foreword to The Global Dynamics of Cellular Automata [16], available at `http://www.cogs.susx.ac.uk/users/andywu/gdca.html`.

[11] Somogyi R and Sniegoski CA (1996) Modeling the complexity of genetic networks: understanding multigene and pleiotropic regulation, Complexity 1, 45-63.

[12] Shmulevich I, Lhdesmki H, Dougherty ER, Astola Zhang JW (2003) The role of certain Post classes in Boolean network models of genetic networks, Proceedings of the National Academy of Sciences of the USA, Vol.100, No.19, 10734–10739.

[13] Wolfram S (1983) Statistical Mechanics of Cellular Automata, Revs. Modern Physics 55(3) 601–664.

[14] Wolfram S (ed) (1986) Theory and Application of Cellular Automata. World Scientific.

[15] Wolfram S (2002) A New Kind of Science, Wolfram Media.

[16] Wuensche A and Lesser MJ (1992) The Global Dynamics of Cellular Automata. Santa Fe Institure studies in the sciences of Complexity, Reference Vol 1, Addison-Wesley, Reading, MA. `http://www.cogs.susx.ac.uk/users/andywu/gdca.html`

[17] Wuensche A (1994) Complexity in One-D Cellular Automata: Gliders, Basins of Attraction and the Z parameter, Santa Fe Institute working paper 94-04-025.

[18] Wuensche A (1994) The ghost in the machine: Basin of attraction fields of random Boolean networks. In: Artificial Life III, ed. Langton CG, Addison-Wesley, Reading, MA, 496–501.

[19] Wuensche A (1995) Discrete Dynamics Lab: Cellular Automata – Random Boolean Networks (DDLab DOS manual).

[20] Wuensche A (1996) The Emergence of Memory: Categorisation Far From Equilibrium, in Towards a Science of Consciousness: The First Tuscon Discussions and Debates, eds. Hameroff SR, Kaszniak AW and Scott AC, MIT Press, Cambridge, MA, 383–392.

[21] Wuensche A (1997) Attractor basins of discrete networks: Implications on self-organisation and memory. Cognitive Science Research Paper 461, DPhil Thesis, University of Sussex.

[22] Wuensche A (1998) Genomic Regulation Modeled as a Network with Basins of Attraction, in Pacific Symposium on Biocomputing '98, eds. Altman RB, Dunker AK, Hunter, Klien TE, World Scientific, Singapore, 89–102.

[23] Wuensche A (1999) Classifying cellular automata automatically: Finding gliders, filtering, and relating space-time patterns, attractor basins, and the Z parameter. Complexity Vol 4/No. 3, 47–66.

[24] Wuensche A (2000) Basins of Attraction in Cellular Automata; Order-Complexity-Chaos in Small Universes, Complexity, Vol.5/no.6, 19–25, based on Complexity in Small Universes, in an art exhibition and events Objective Wonder: Data as Art, and Merged Realities Exposition'99, Univ of Arizona, in collaboration with Chris Langton, details at `http://www.cogs.susx.ac.uk/users/andywu/Exh2/Exh3.html`.

[25] Wuensche A (2001) Discrete Dynamics Lab: the DDLab web site with the software, manual and other resources: `http://www.ddlab.org`
and `http://www.cogs.susx.ac.uk/users/andywu/ddlab.html`.

[26] Wuensche A (2005) Glider dynamics in 3-value hexagonal cellular automata: the beehive rule, Int. Journ. of Unconventional Computing, Vol.1, No.4, 2005, 375–398.

[27] Wuensche A and Adamatzky A (2006) On spiral glider-guns in hexagonal cellular automata: activator-inhibitor paradigm, International Journal of Modern Physics C,Vol. 17, No. 7,1009–1026.

[28] Wuensche A (2008) Encryption using cellular automata chain-rules, Automata-2008 Theory and Applications of Cellular Automata, Luniver Press, 126-136.