Visualising the dynamics of Random Boolean Networks:

examples of network size, mutation, canalisation

Susan Stepney

University of York Technical Report YCS-2010-448

18 February 2010

Abstract

We propose a simple approach to visualising the time behaviour of Random Boolean Networks (RBNs), and demonstrate the approach in a variety of cases: examining the dynamics as a function of network size for $K = 2$ connectivity networks, examining the effect of state and structure mutations on $K = 2$ networks, and examining the effect of canalising functions for $K > 2$ networks. We provide a Matlab implementation of the visualisation algorithm, and of the various demonstrations.

Contents

Chapter 1

Introduction

Random Boolean networks (RBNs) are a well-studied form of complex discrete dynamical systems [1, 2, 3, 4, 9]. Visualisation of the dynamics can aid understanding, but (unlike for 1D Cellular Automata, for example), there has been no satisfactory visualisation of RBN time behaviour. In [5] we propose a simple approach to visualising the time behaviour of RBNs; in [6] we illustrate its use for examining network mutations and canalisation. Here we demonstrate the approach in a variety of cases: examining the dynamics as a function of network size for $K = 2$ networks, examining in detail the effect of state and structure mutations for $K = 2$ networks, and examining in detail the effect of canalising functions for $K > 2$ networks. We also provide Matlab code (in the Appendix) for these visualisations and mutations.

Chapter 2

RBNs

2.1 Definitions

A Random Boolean Network (RBN) [3, 4] comprises N *nodes*. Each node i at time t has a binary valued state, $s_{i,t} \in \mathcal{B}$. Each node has K inputs assigned randomly from K of the N nodes (an input may be from the node itself)[1]; the wiring pattern is fixed throughout the lifetime of the network. This wiring defines the node's neighbourhood, $\nu_i \in N^K$. See figure 2.1.

The state of node i's neighbourhood at time t is $\chi_{i,t} \in \mathcal{B}^K$, a K-tuple of node states that is the projection of the full state onto the neighbourhood ν_i.

Each node has its own randomly chosen local state transition rule, or update rule, $\phi_i : \mathcal{B}^K \to \mathcal{B}$. These nodes form a network of state transition machines. At each timestep, the state of each node is updated in parallel, $s_{i,t+1} = \phi_i(\chi_{i,t})$.

The global dynamics f is determined by the local rules ϕ_i and the connectivity pattern of the nodes ν_i.

Given a particular global state $\mathbf{s}_0 \in \mathcal{B}^N$, its trajectory under f is a sequence of states $\mathbf{s}_0, \mathbf{s}_1, \ldots, \mathbf{s}_t, \ldots$. Eventually, because the state space is finite, a state that was met before will be met again: there exists a k such that $\mathbf{s}_k = \mathbf{s}_{k+p}$, for some p. Since the dynamics is deterministic, the trajectory will then recur: for all $i \geq k . \mathbf{s}_i = \mathbf{s}_{i+p}$. The system has entered an *attractor*, with cycle length or period p. States not on an attractor are called *transient*.

The set of all states $\mathbf{s}_j$ whose trajectories lead to the same attractor forms the *basin of attraction* of that attractor. The total state space is partitioned into these basins: every state is in precisely one basin.

[1] These wiring conditions are not stated explicitly by Kauffman [3, 4]. However, in the $K = N$ case, Kauffman [4, p.192] states that "Since each element receives an input from all other elements, there is only one possible wiring diagram". This implies that multiple connections from a single node are not allowed in a true RBN (otherwise more wiring diagrams would be possible) whereas self connections are allowed (otherwise K would be restricted to a maximum value of $N-1$). Subsequent definitions (for example [1, §2.A]) explicitly use the same conditions as given here.

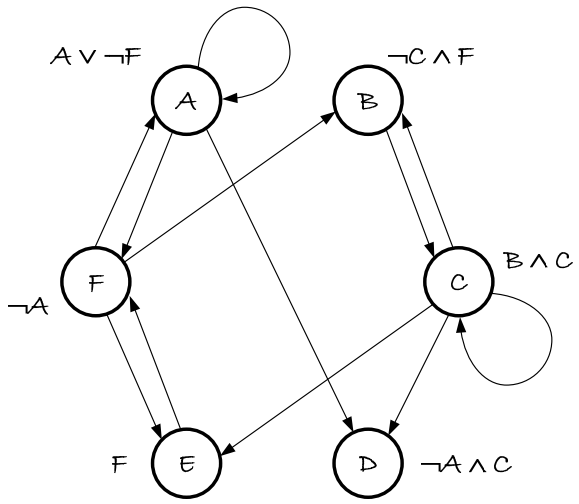

Figure 2.1 An example RBN with $N = 6, K = 2$. Each node has $K = 2$ inputs; it can have any number of outputs. So the neighbourhood function is $\nu_A = (A, F), \nu_B = (C, F)$, etc. Each node combines its inputs by a random boolean function ϕ_i: that function might ignore one or more of the inputs.

K	cycle length	#attractors	stability	reachability
1	$O(\sqrt{N})$	$O(2^N)$	low	high
2	$O(\sqrt{N})$	$O(\sqrt{N})$	high	low
> 5	$O(2^N)$	$O(N)$	low	high

Table 2.1 Dynamics of RBNs for different K, adapted from [4, table 5.1]. Stability and reachability are discussed in §5.1.

2.2 Ordered dynamics

Kauffman [3, 4] investigates the properties of RBNs as a function of connectivity K. Despite all their randomness, "such networks can exhibit powerfully ordered dynamics" [3], particularly when $K = 2$ (table 2.1). Kauffman investigates RBNs as simplified models of gene regulatory networks (GRNs). He notes that "cell types are constrained and apparently stable recurrent patterns of gene expression", and interprets his RBN results as demonstrating that a "cell type corresponds to a state cycle attractor" [4, p.467] (in a $K = 2$ network).

Drossel [1, §1] notes that subsequent computer simulation of much larger networks shows that "for larger N the apparent square-root law [of attractor numbers

and cycle lengths] does not hold any more, but that the increase with system size is faster". This faster growth is dominated by the appearance of a few complex "relevant components" (the parts of the network that are not determined by the values of other nodes, and so determine the attractors) with exponentially long attractors, which "dominate the mean attractor [cycle] length". The asymptotic behaviour is seen only for "networks with more than 100000 nodes", significantly larger than those visualised here ($N = 100 - 800$).

2.3 Frozen core

Kauffman [4, p.203] observes that $K = 2$ RBNs "develop a connected mesh, or *frozen core*, of elements, each frozen in either the 1 or 0 state." Some nodes may freeze in one attractor, but not in another: the frozen core itself comprises the "nodes that are frozen on the same value on all attractors" [1, §5].

Drossel [1, §5.A] discusses the structure of the frozen core in some detail. It arises in two ways. Firstly, "there are constant functions that fix the values of some nodes, which in turn lead to the fixation of the values of some other nodes, etc." Secondly, there are "self-freezing loops", where nodes remain in a certain state once they have entered it, and, for large networks, the chance of never entering the self-freezing state is low.

Chapter 3

Visualising the dynamics

3.1 Introduction

Good visualisations can aid the understanding of complex systems, and can help generate new questions and hypotheses about their behaviours.

3.2 Attractor basins

Visualising the global structure of boolean networks (RBNs and Cellular Automata) can help in understanding some aspects of their dynamics. For small systems, a common approach is to lay out the global state transition graph (defined by f) to highlight the separate basins and their attractors (see figure 3.1).

Wolfram [8, fig 9.1] used this approach in early work on cellular automata; Wuensche [10, 9] has developed special purpose layout software, and uses this approach consistently, to highlight aspects of the dynamics. This state transition graph approach emphasises global state structure, and de-emphasises the states of the individual nodes. It also does not scale particularly well, since the full transition graph has 2^N states.

3.3 Time dynamics of CAs

Another approach is commonly used to visualise the time-dependent states of the nodes. For 1D cellular automata (CAs), the global behaviour from a given initial state is conventionally visualised by drawing the global state at time t as a line of nodes (with colours corresponding to the local state), then drawing the state at $t + 1$ directly below, and so on (see figure 3.2). This approach highlights the propagation of information through the CA.

CAs have a regular topology, which is used when laying out the nodes for vi-

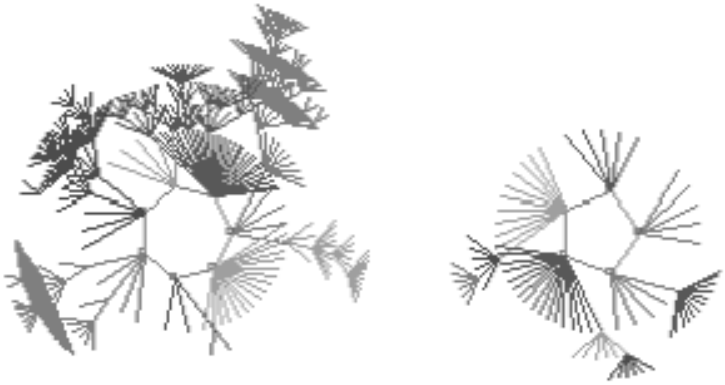

Figure 3.1 Visualisation of two basins of attraction of an RBN, from [9, fig.4]

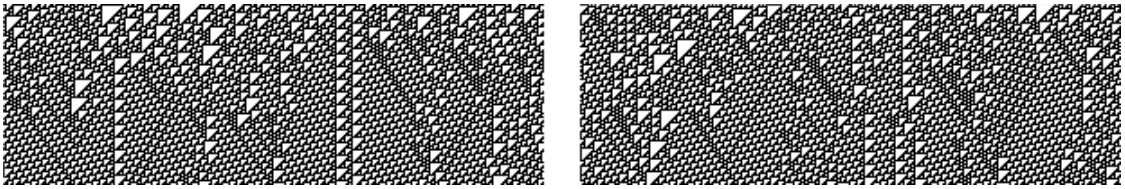

Figure 3.2 Visualisation of the time evolution of ECA rule 110, with $N = 300$, 100 timesteps, and two different random (50% "on", 50% "off") initial conditions

sualisation. RBNs have no such regular topology. If this approach is taken with their nodes laid out at random (as done, for example, in [9, fig.3] or [2, fig.2]), the structure of the dynamics is hard to discern: see figure 3.3.

3.4 Using the frozen core

We use the existence of a frozen core (§2.3) to provide an order for placing the nodes in the visualisation. Nodes frozen in the 1 or 0 state are placed towards the edges of the figure; nodes that are changing state are placed towards the centre: see figure 3.4. The different transient behaviours and attractors are now clearly visible; for example, it is clear that these show three different attractors, with three different periods.

The algorithm used here is as follows (see appendix B.1 for Matlab source code). For a given RBN, to determine the order of drawing the nodes in the visualisation, do the following: (1) Pick a representative number of timesteps, t (for example,

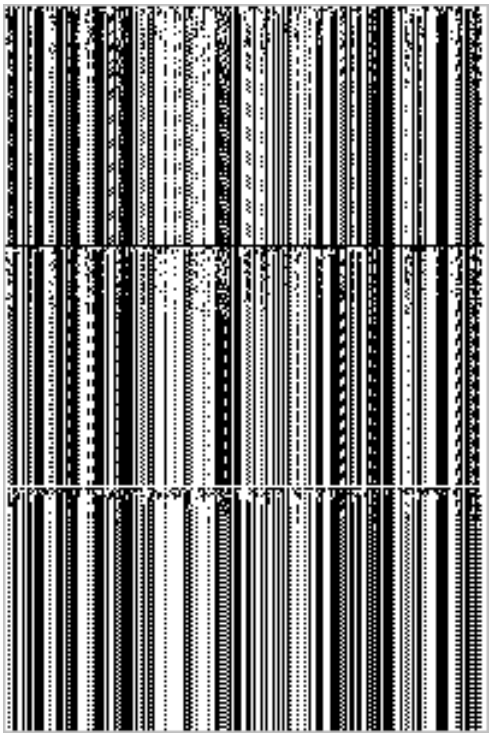

Figure 3.3 Visualisation of the time evolution of a typical $K = 2$ RBN, with $N = 200$, and initial condition all nodes randomised (50% "on", 50% "off"); reinitialised after 100 timesteps with all nodes "on"; reinitialised after a further 100 timesteps with all nodes "off".

the number that will be used in the subsequent visualisations). (2) Set the RBN into a given initial state (here, random). (3) Run it for t timesteps, counting how many times each node is on. Repeat steps 2 and 3 for other suitable conditions, accumulating the counts. (4) Sort the nodes by the total number of times they were on in these runs.

This has the effect that the frozen core nodes move to the edges of the figures, since they are in a constant state (after transient behaviour has died out), whilst the nodes with cycling states are in the centre. Additionally, the frozen core nodes with shorter transient behaviour will be closer to the edges than those with longer transient behaviours. Similarly, nodes with cycling states will be sorted according to the amount of time they spend in one state or the other, with those half the time in each state towards the centre. This tends to highlight the attractor structure.

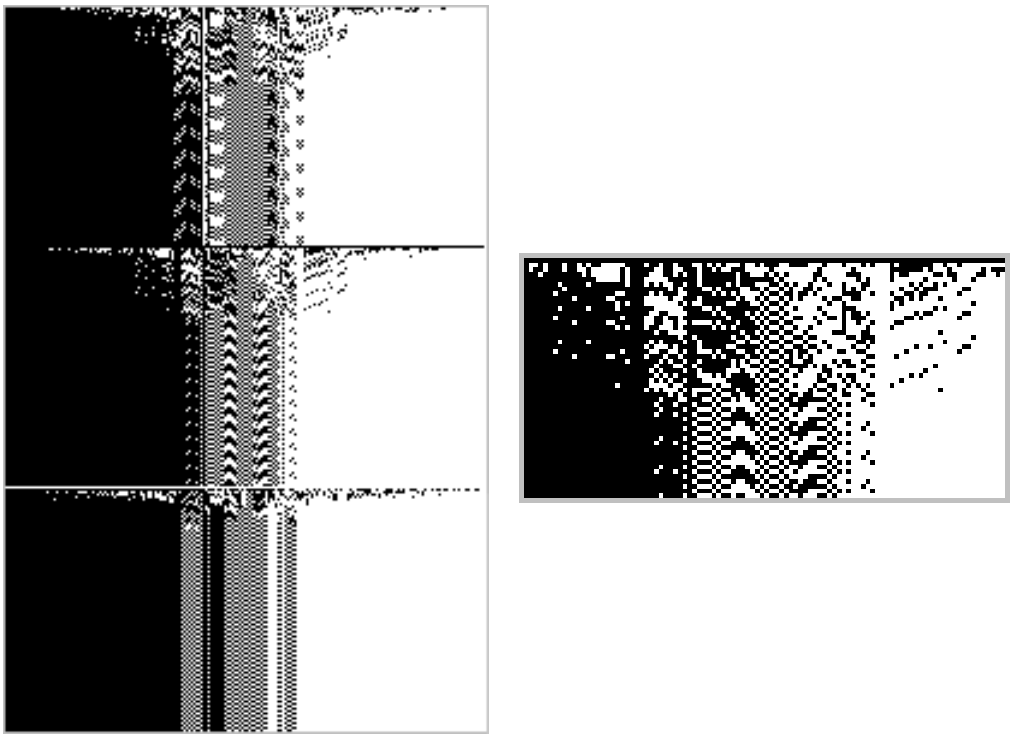

Figure 3.4 Visualisation of the time evolution of the $K = 2$ RBN shown in figure 3.3, (a) with the nodes sorted to expose the frozen core; (b) zooming in on the structure of the cycling region.

3.5 Effect of reordering

Clearly, nodes with boolean function 0 (false) and 15 (true) will be sorted to the extremes: these nodes are off (on) all the time, and so will have minimal (maximal) count. What of the other nodes? Figure 3.5 shows that these are biassed: boolean functions with one on output tend to shift to the left and those with three on outputs tend to shift to the right (into the frozen core); those with two on outputs are clustered more to the centre (the active region).

This bias is not very strong, though, and sorting on boolean function alone is insufficient to reveal the attractor structure (figure 3.6).

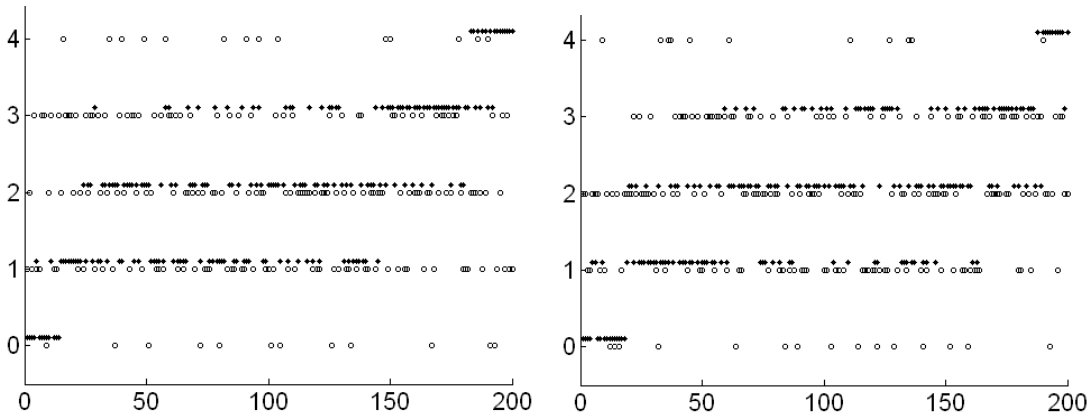

Figure 3.5 Scatter plot of the boolean function distribution of $K = 2$ RBNs with $N = 200$. Open circles are (random) positions before sorting; filled circles are (biassed) positions after sorting. y-axis value is the number of inputs to the boolean function that give an output of 1. (See appendix B.2 for Matlab source code).

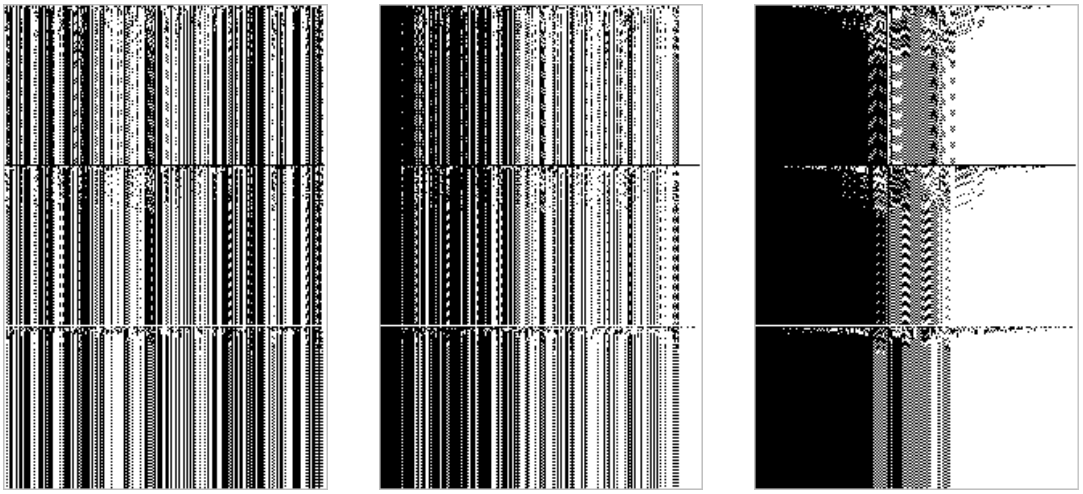

Figure 3.6 Visualisation of the time evolution of a typical $K = 2$ RBN, with $N = 200$, and initial condition all nodes randomised (50% "on", 50% "off"); reinitialised after 100 timesteps with all nodes "on"; reinitialised after a further 100 timesteps with all nodes "off". (a) left: unsorted (as figure 3.3); (b) middle: sorted on boolean function value; (c) right: sorted on frozen core activity (as figure 3.4)

3.6 How many runs to sort

Note that the precise order depends on the various initial states chosen, and how many different runs are sorted.

For reproducibility, the network could be run only from initial states that can be reconstructed independent of the node ordering (for example, all zeroes, all ones, or some non-trivial some function of the node's boolean operation). Here, however, for simplicity, we run from random initial states (with an average of half nodes on, half off). Choosing reproducible states, or others proportions of nodes on and off, makes no observable difference.

Figures 3.7 shows a $K = 2$, $N = 200$ RBN. The RBN is run 10 times, from 10 different initial conditions. The columns show the results when the nodes are sorted on the first 1, 2, 4, and 8 runs. When sorted on only the first run, the frozen core is not well-identified. When sorted on only two runs, the core is fairly well identified, but the transient behaviour looks a little scrappy. Sorting on 4 runs exposes most of the behaviour; sorting on 8 runs does not seem to expose much more structure.

Figure 3.8 shows a different $K = 2$, $N = 200$ RBN; much the same observations apply.

Figures 3.9–3.12 show a $K = 2$, $N = 800$ RBN with the nodes sorted on the first 1, 2, 4, and 6 runs. Again, sorting on the first 4 runs seems to expose the structure sufficiently.

For a more quantitative analysis, the number of runs needed to expose the structure could be investigated by calculating the correlation between the ordering after different number of runs. But for the visualisations here, in all subsequent examples, we simply sort the nodes on 4 runs. Note that sorting on more runs gives different detail, but the gross overall structure is unchanged: transient lengths, cycle length, etc.

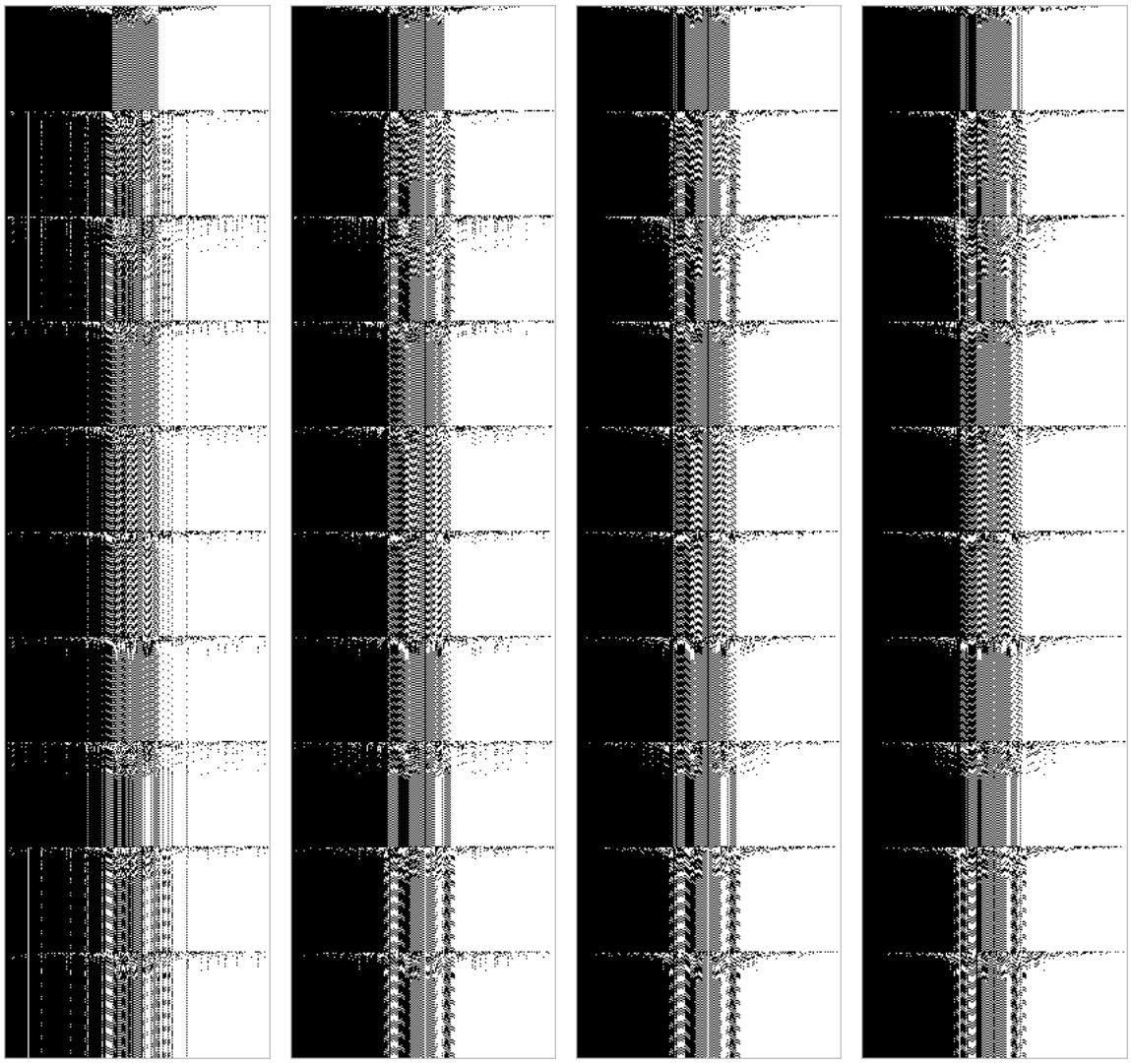

Figure 3.7 Visualisation of the time evolution of a typical $K = 2$ RBN, with $N = 200$, $t = 80$. The nodes are sorted on the first 1, 2, 4, 8 runs.

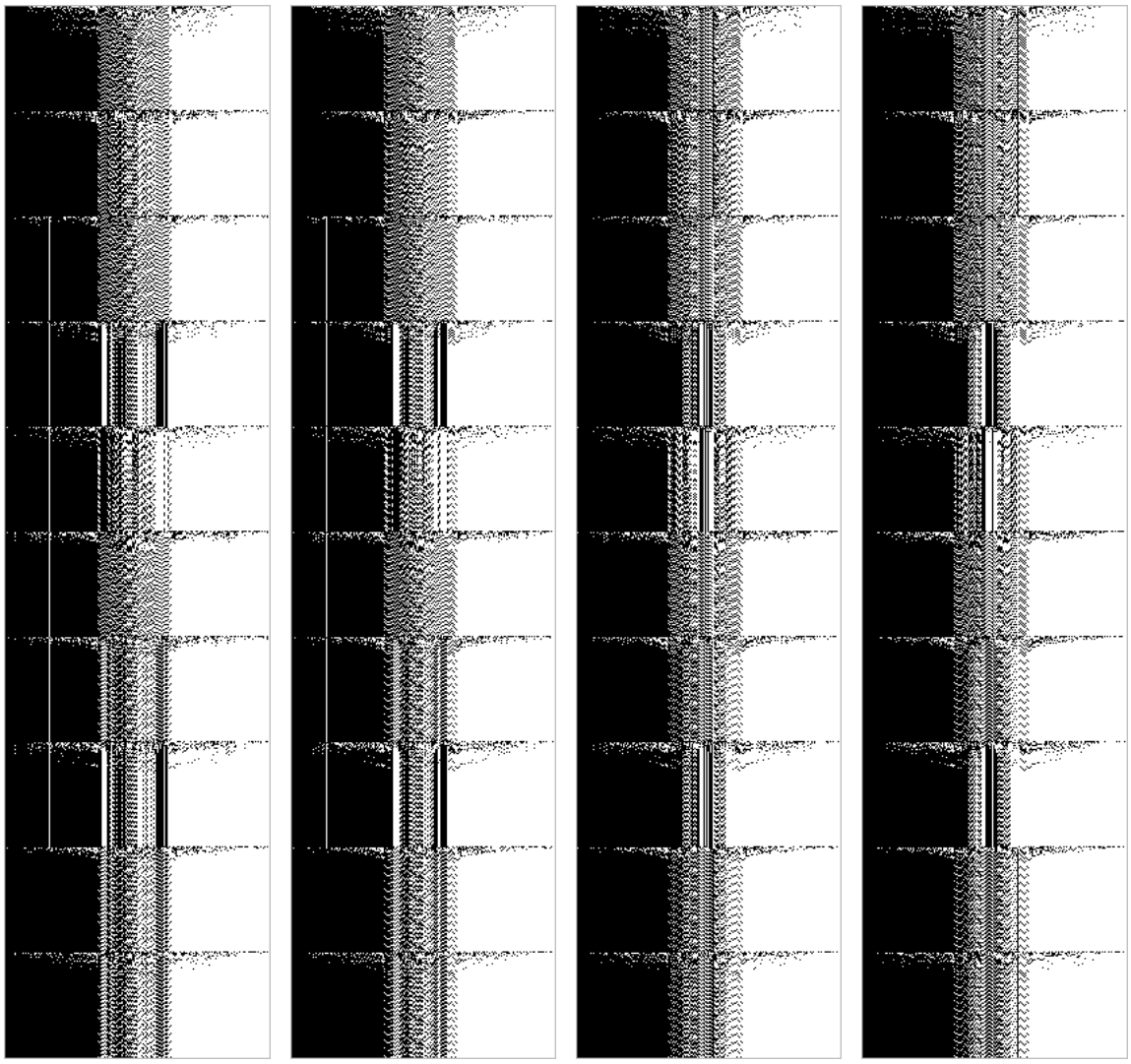

Figure 3.8 Visualisation of the time evolution of another typical $K = 2$ RBN, with $N = 200$, $t = 80$. The nodes are sorted on the first 1, 2, 4, 8 runs.

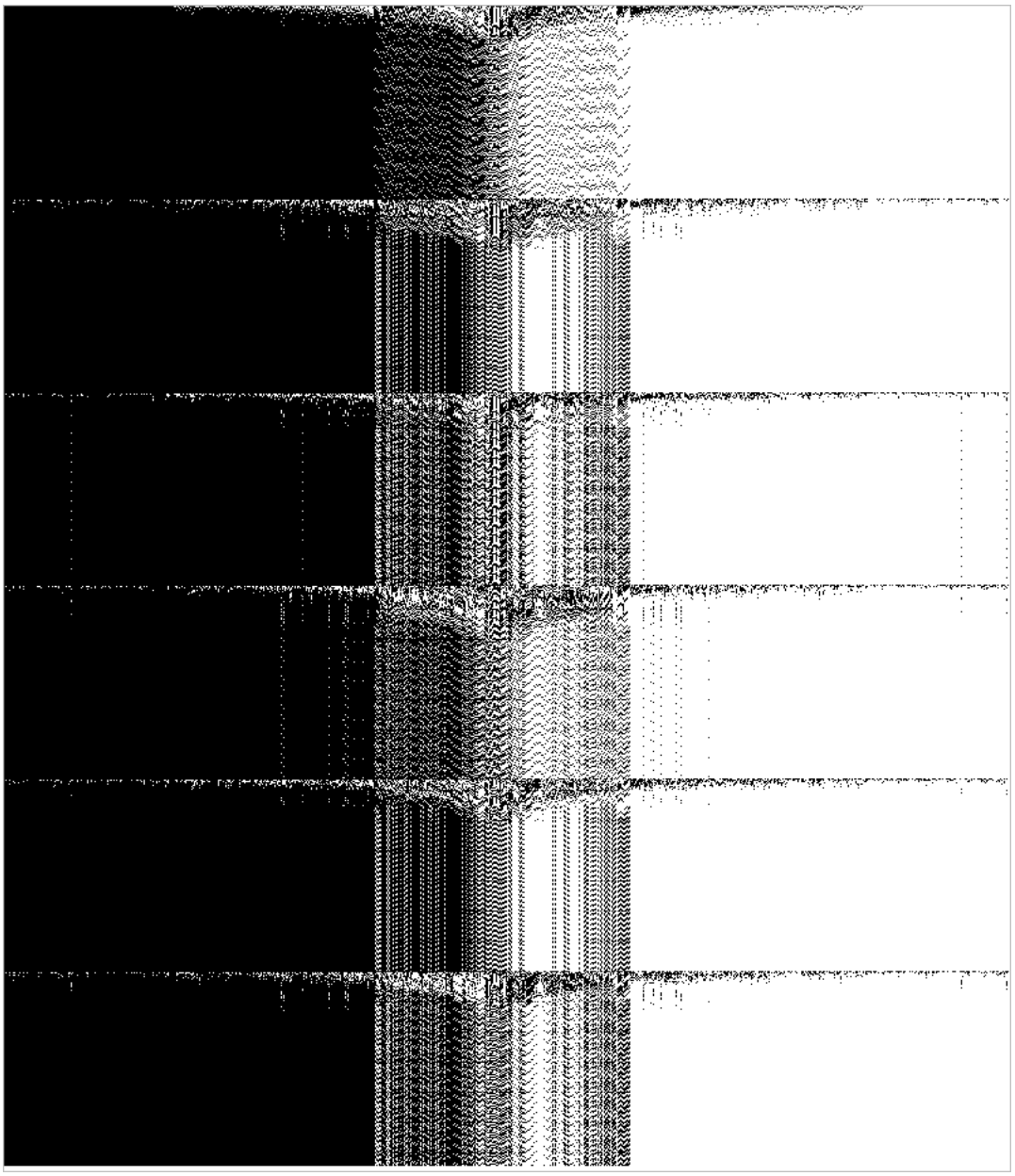

Figure 3.9 Visualisation of the time evolution of a typical $K = 2$ RBN, with $N = 800$, $t = 160$. The nodes are sorted on the first run.

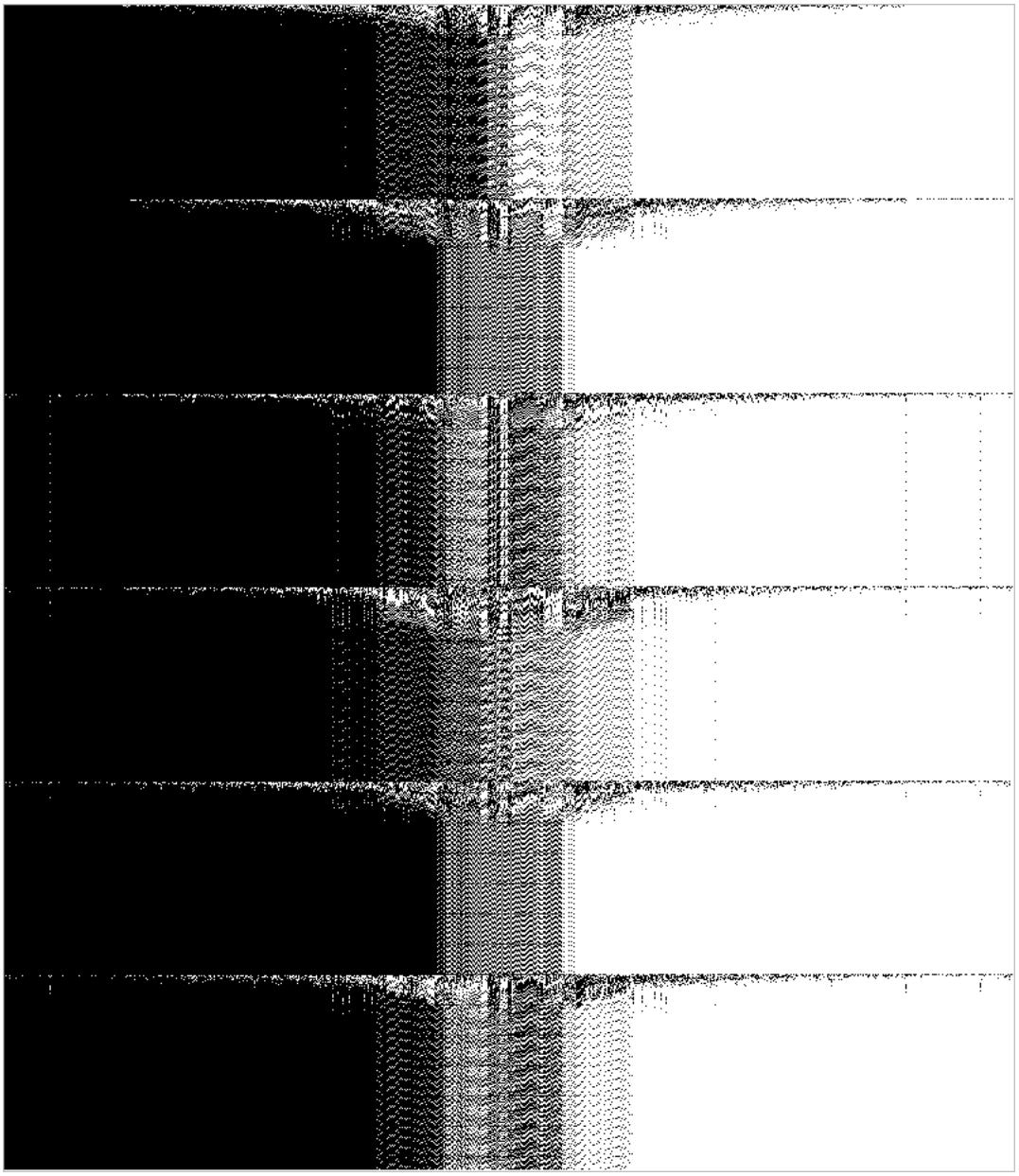

Figure 3.10 Visualisation of the time evolution of a typical $K = 2$ RBN, with $N = 800$, $t = 160$. The nodes are sorted on the first 2 runs.

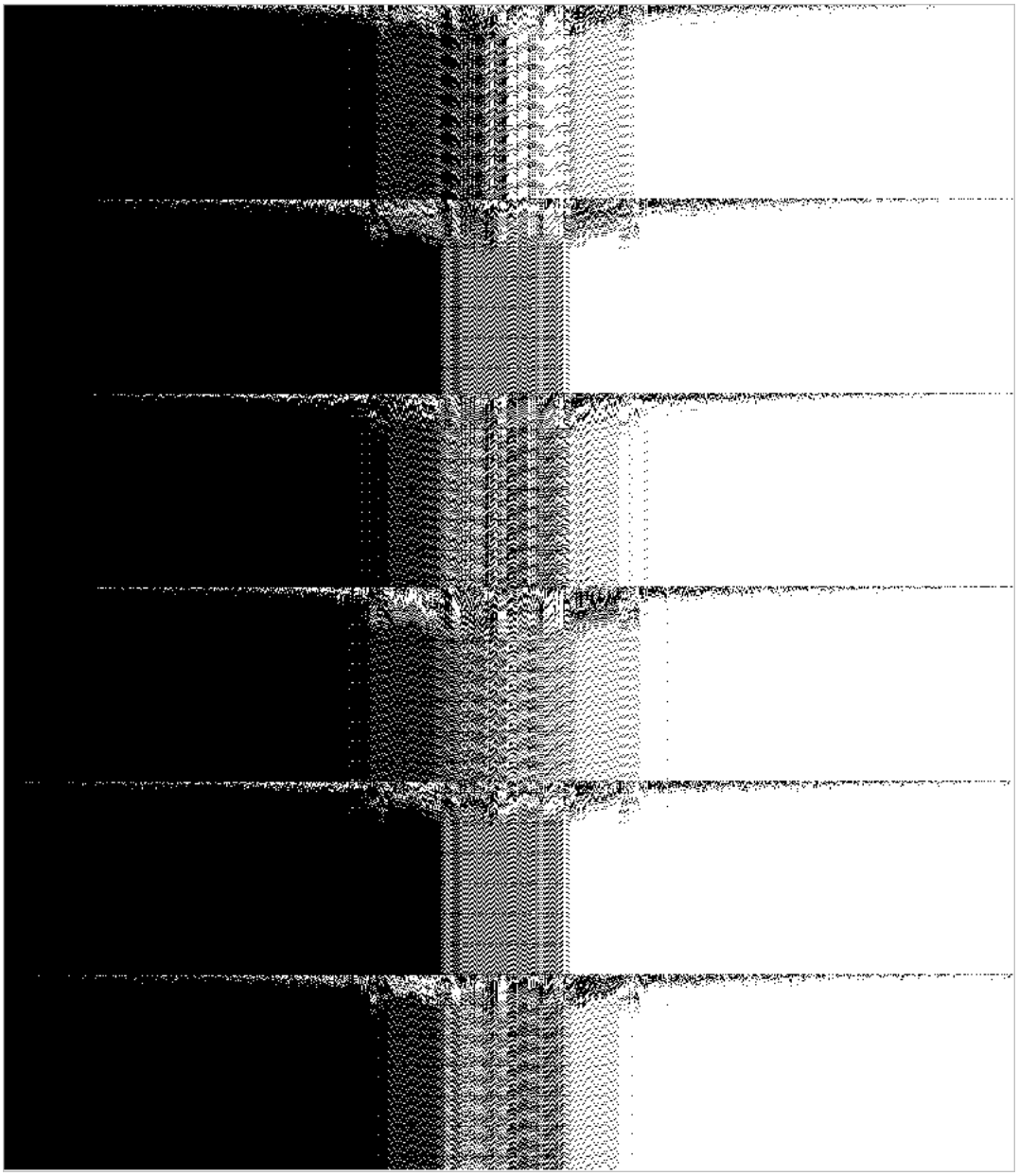

Figure 3.11 Visualisation of the time evolution of a typical $K = 2$ RBN, with $N = 800$, $t = 160$. The nodes are sorted on the first 4 runs.

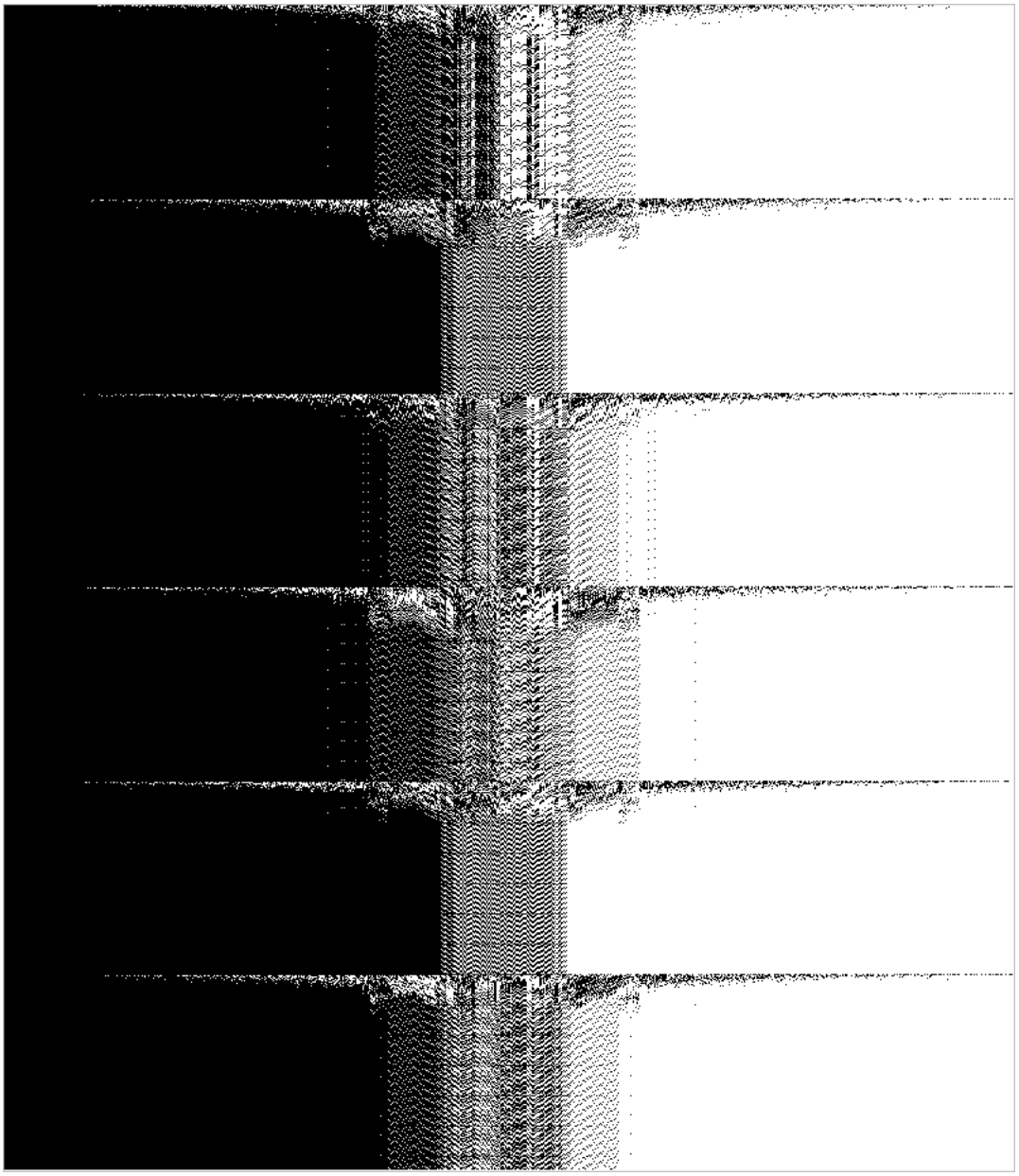

Figure 3.12 Visualisation of the time evolution of a typical $K = 2$ RBN, with $N = 800$, $t = 160$. The nodes are sorted on the first 6 runs.

3.7 How many timesteps to sort

The precise order also depends on the length of the run (number of timesteps) used.

Figure 3.13 shows a $K = 2$, $N = 200$ RBN, sorted on different numbers of timesteps. Clearly, $t = 20$ is insufficient; the transient behaviour is hiding the active node behaviour. There is a small difference between 50 and 100 timesteps, but this is the same order of difference as between sorting on 4 and 8 runs (last column).

So a good heuristic is to make the length of the run at least twice the length of the longest transient, which is also a good choice for the minimum length to display the structure of the dynamics. For a more quantitative analysis, the number of timesteps needed could be investigated by calculating the correlation between orderings using different number of timesteps.

In all subsequent examples, unless states otherwise, we sort the nodes on the total length of each run displayed.

3.8 Examples

In the following chapters, we explore several different aspects of RBNs, using this visualisation approach to expose the relevant features.

We use Tufte's "small multiples" [7] technique, which "allows the viewer to focus on changes in the data", by displaying an array of RBNs that can be readily compared.

The aim is to use the visualisation to prime intuition and aid understanding of RBNs' rich dynamics, and to provoke hypotheses about the detailed behaviours. Any such hypotheses would need to be investigated in a rigorous manner.

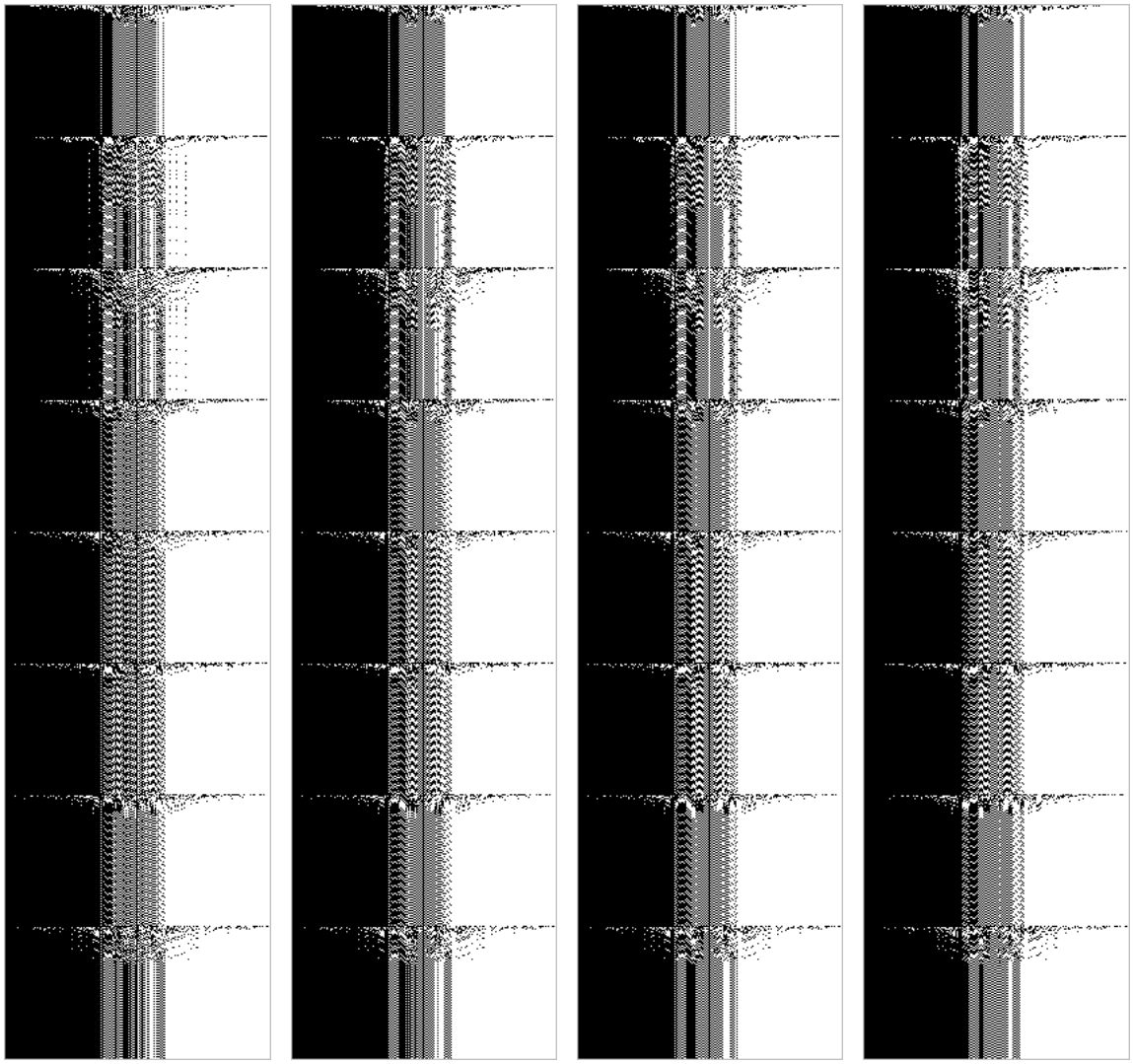

Figure 3.13 Visualisation of the time evolution of a typical $K = 2$ RBN, with $N = 200$, $t = 100$. In the first three columns, the nodes are sorted on the first 4 runs, and the first 20, 50, 100 timesteps. In the last column, the nodes are sorted on all 8 runs, and the first 100 timesteps.

Chapter 4

Network size

Here we visualise the effect of the number of nodes, N, on the time behaviour of $K = 2$ networks. As Kauffman notes (table 2.1), for $K = 2$ RBNs, the (mean) attractor cycle length and number of attractors both go as $O(\sqrt{N})$. Here we show visualisations of $K = 2$ RBNs with different N, ranging from $N = 100$ ($\sqrt{N} = 10$), to $N = 800$ ($\sqrt{N} \approx 28$): see figures 4.1–4.9.

These visualisations demonstrate that there are indeed short period attractors. But they also suggest further possible properties: (a) the number of cycling nodes involved in the attractors does not vary widely within a single RBN, although it does varies widely across RBNs with the same N; (b) the frozen core within an RBN is well conserved[1] (take together, these imply that the nodes that are cycling or frozen in one attractor basin are likely to be similarly cycling or frozen in all attractor basins); (c) the frozen core is balanced: there are roughly the same number of nodes frozen on as frozen off; (d) the size of the frozen core is typically more than half the nodes[2]; (e) the transient behaviour of the frozen core is well conserved in a given RBN: frozen nodes tend to have short or long transient behaviours independent of the attractor; (f) the length of the transient behaviour of the frozen core does not depend strongly on N (although the transient behaviour of the cycling nodes does, probably because there are potentially many more cycling nodes in large N RBNs).

More such conjectures could be generated from larger numbers of examples; some of these may be worthy of further investigation. In particular, what these visualisations help to show is the scale of the variation between attractors in one RBN, and the much greater variation between RBNs. Analytic calculations and numerical experiments tend to focus on mean behaviour, rather than on variation.

[1] Strictly, the frozen core comprises the "nodes that are frozen on the same value on all attractors" [1, §5]. However, there are some nodes that are frozen on one attractor, but not on another. The observation here is that there tend not to be many of these latter "sometimes frozen" nodes.

[2] Drossel [1, §5.A] notes that the mean number of nodes *not* in the frozen core is proportional to $N^{2/3}$ for large N.

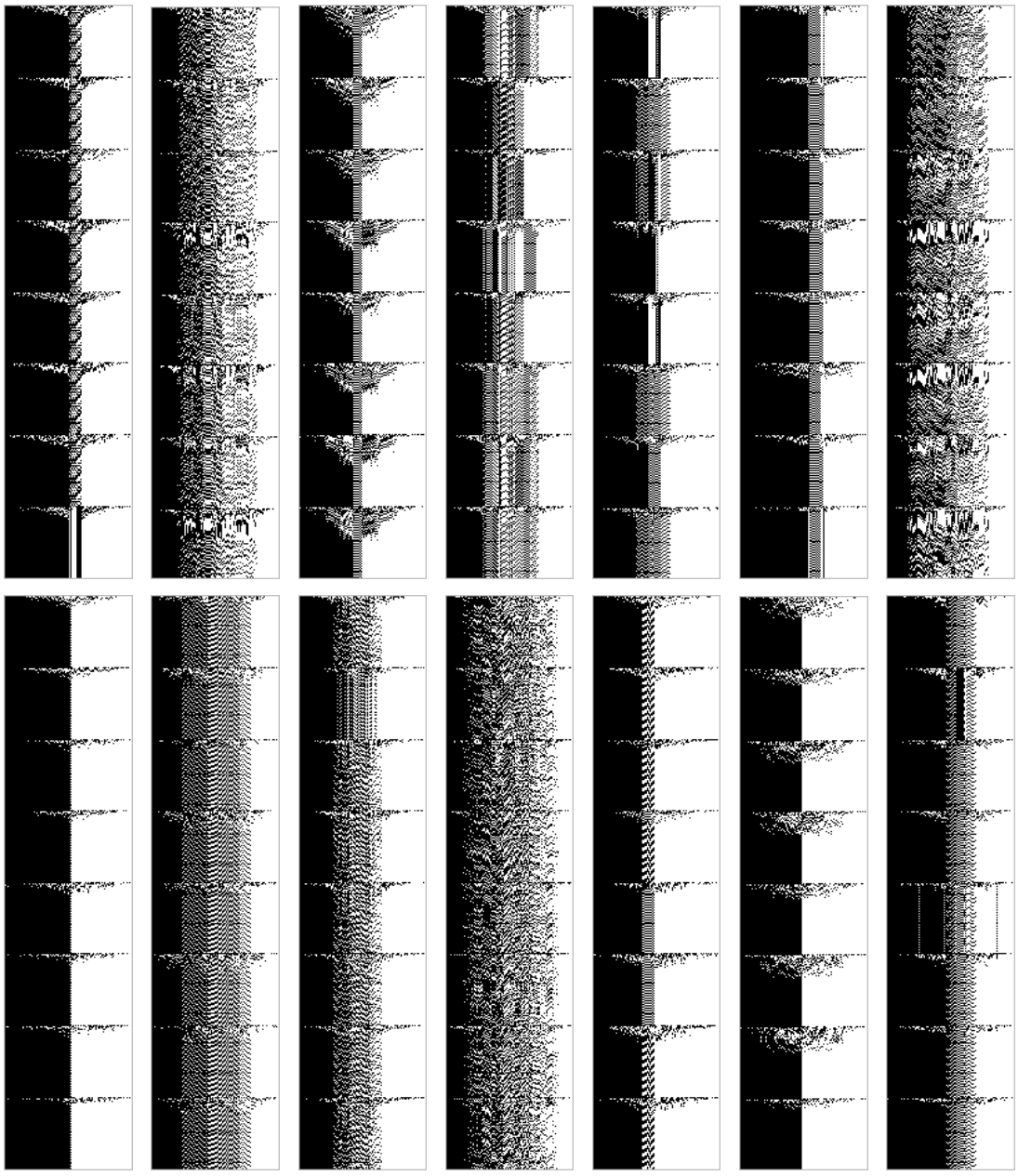

Figure 4.1 Visualisation of the time evolution of 14 typical $K = 2$ RBNs, with $N = 100$. Every 60 timesteps the nodes are reinitialised to a new random configuration, to explore other attractors. They exhibit ordered behaviour: short transients, and low period attractors. In each case, the nodes are sorted over the first four runs: note that this has not been sufficient to isolate the frozen core in the final example (bottom right).

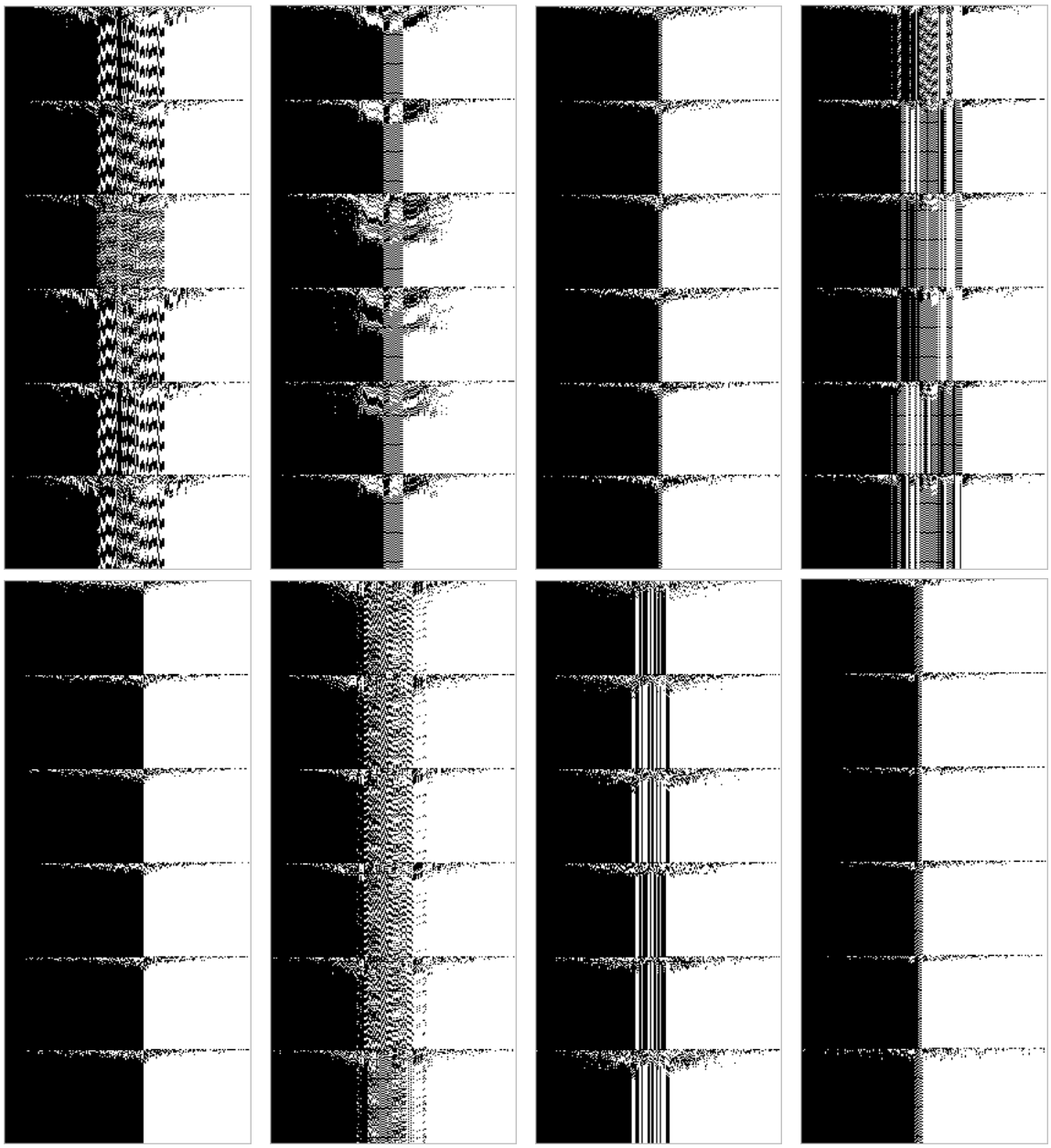

Figure 4.2 Visualisation of the time evolution of eight typical $K = 2$ RBNs, with $N = 200$. Every 80 timesteps the nodes are reinitialised to a new random configuration, to explore other attractors. They exhibit ordered behaviour: short transients, and low period attractors. In each case, the nodes are sorted over the first four runs.

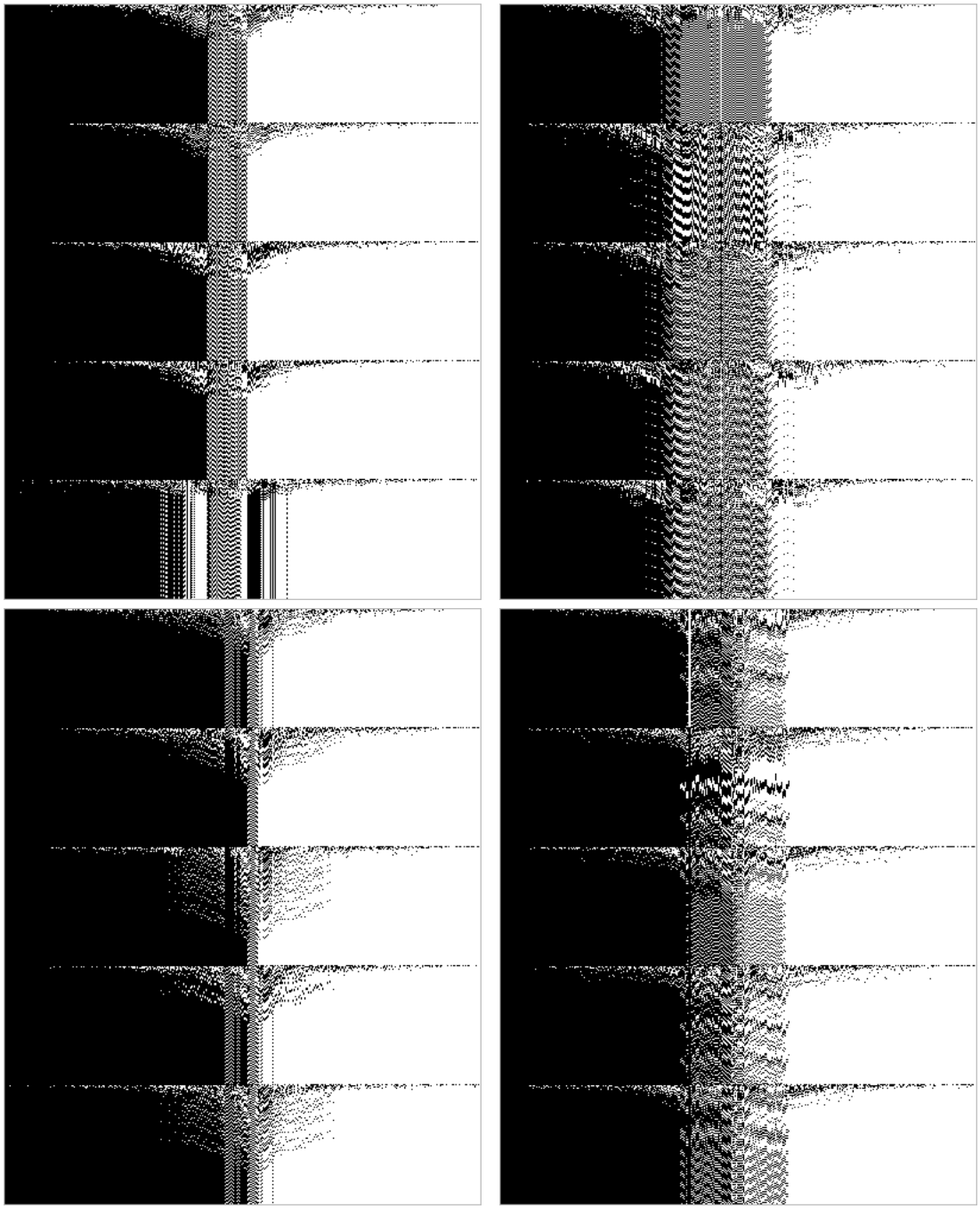

Figure 4.3 Visualisation of the time evolution of four typical $K = 2$ RBNs, with $N = 400$. Every 100 timesteps the nodes are reinitialised to a new random configuration, to explore other attractors. They exhibit ordered behaviour: short transients, and low period attractors.

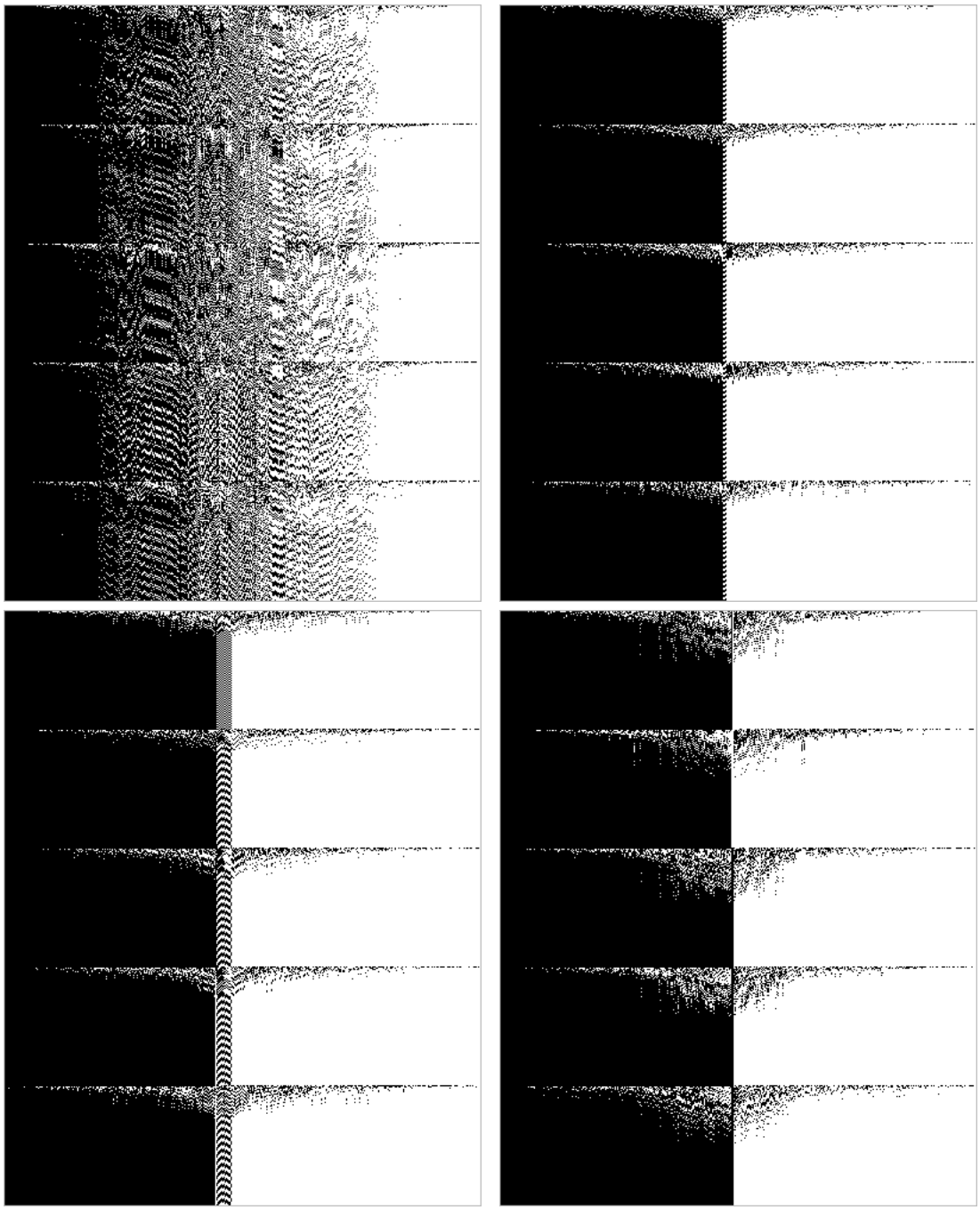

Figure 4.4 Visualisation of the time evolution of four further typical $K = 2$ RBNs, with $N = 400$.

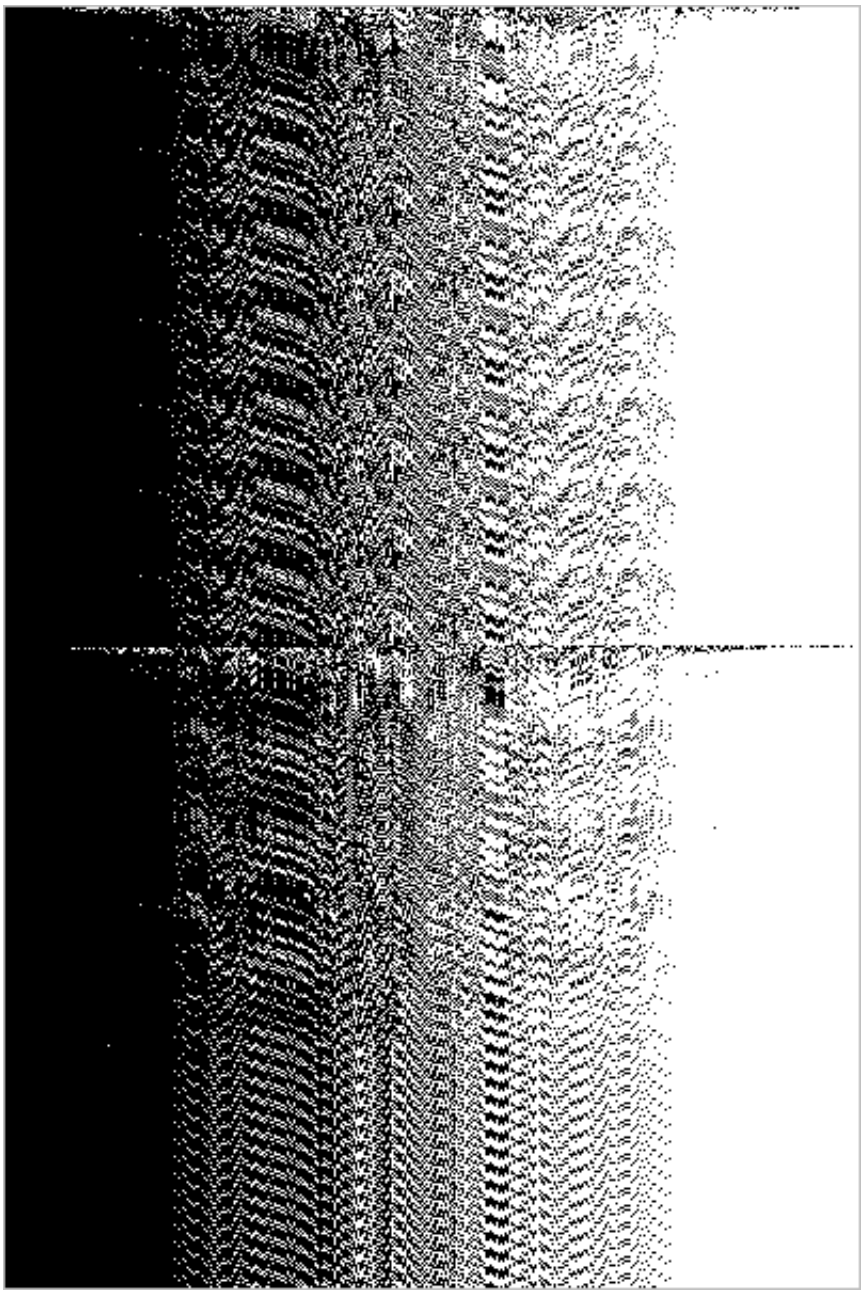

Figure 4.5 Visualisation of the time evolution of the long transient $K = 2$ $N = 400$ RBN, (figure 4.4, top left), with $t = 300$, to show the attractors of the first two runs.

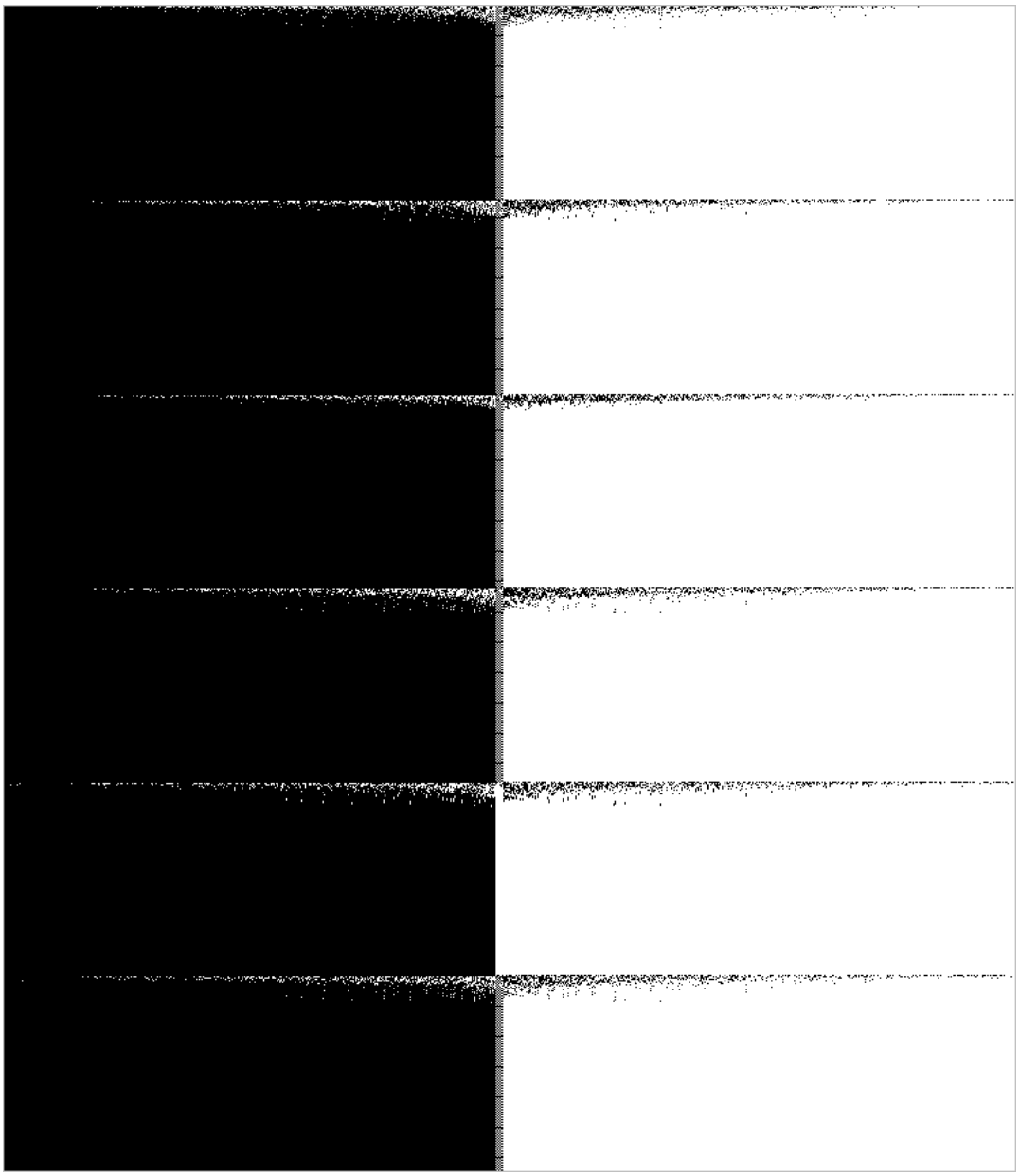

Figure 4.6 Visualisation of the time evolution of a typical $K = 2$ RBN, with $N = 800$. Every 160 timesteps the nodes are reinitialised to a new random configuration, to explore other attractors. It exhibits ordered behaviour: short transients, and low period attractors. The nodes are sorted over the first four runs.

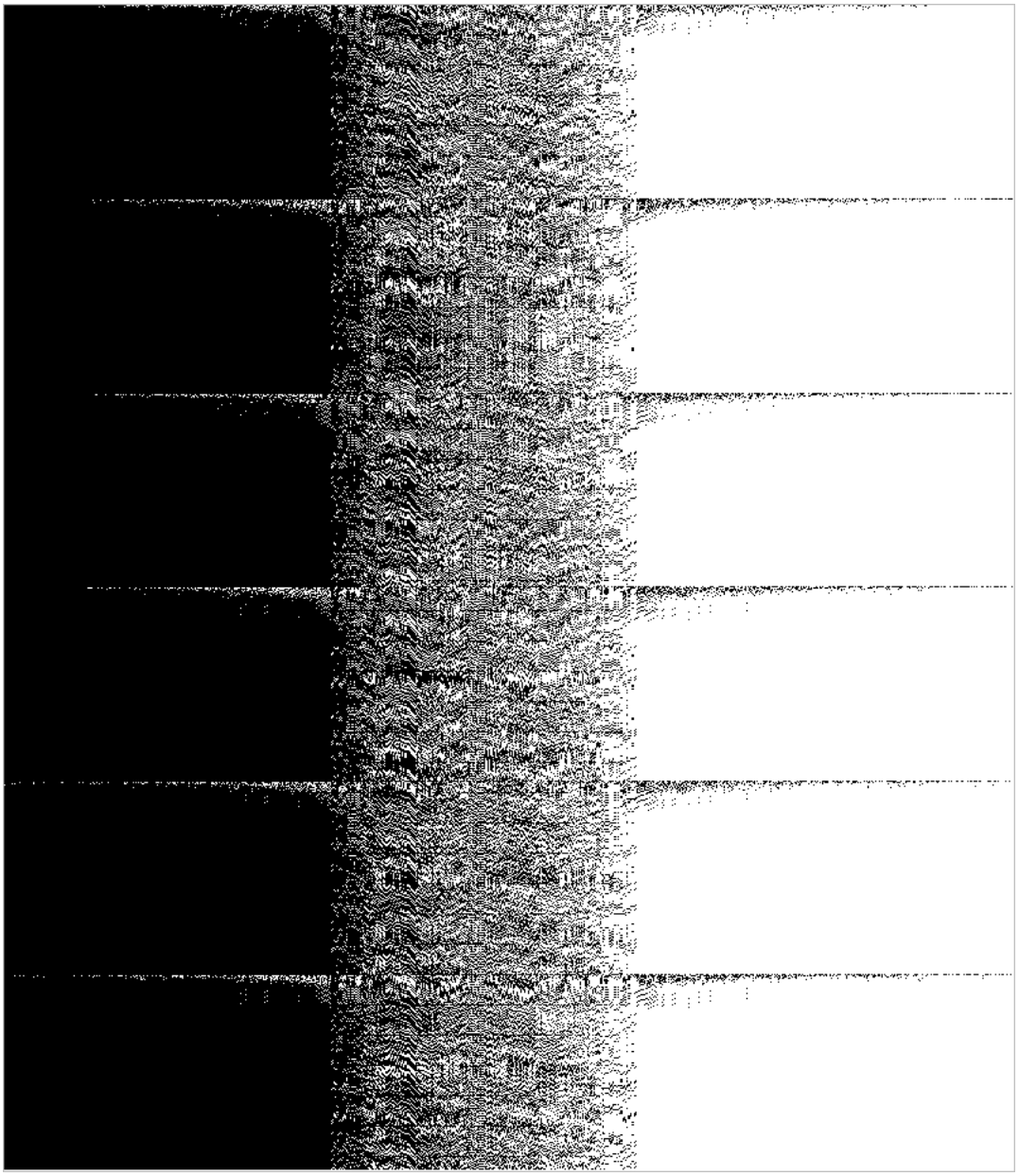

Figure 4.7 Visualisation of the time evolution of a further typical $K = 2$ RBN, with $N = 800$.

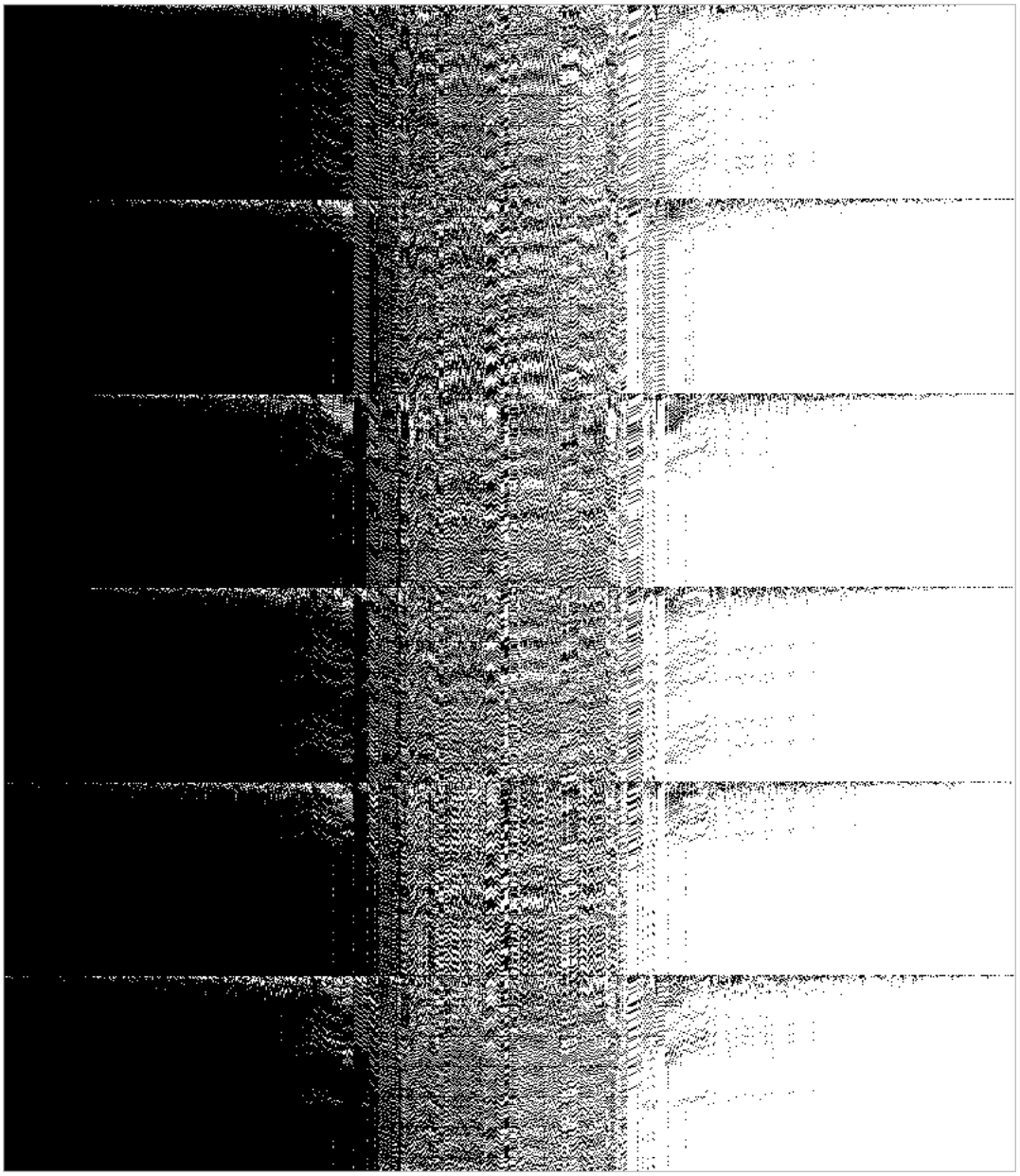

Figure 4.8 Visualisation of the time evolution of a further typical $K = 2$ RBN, with $N = 800$.

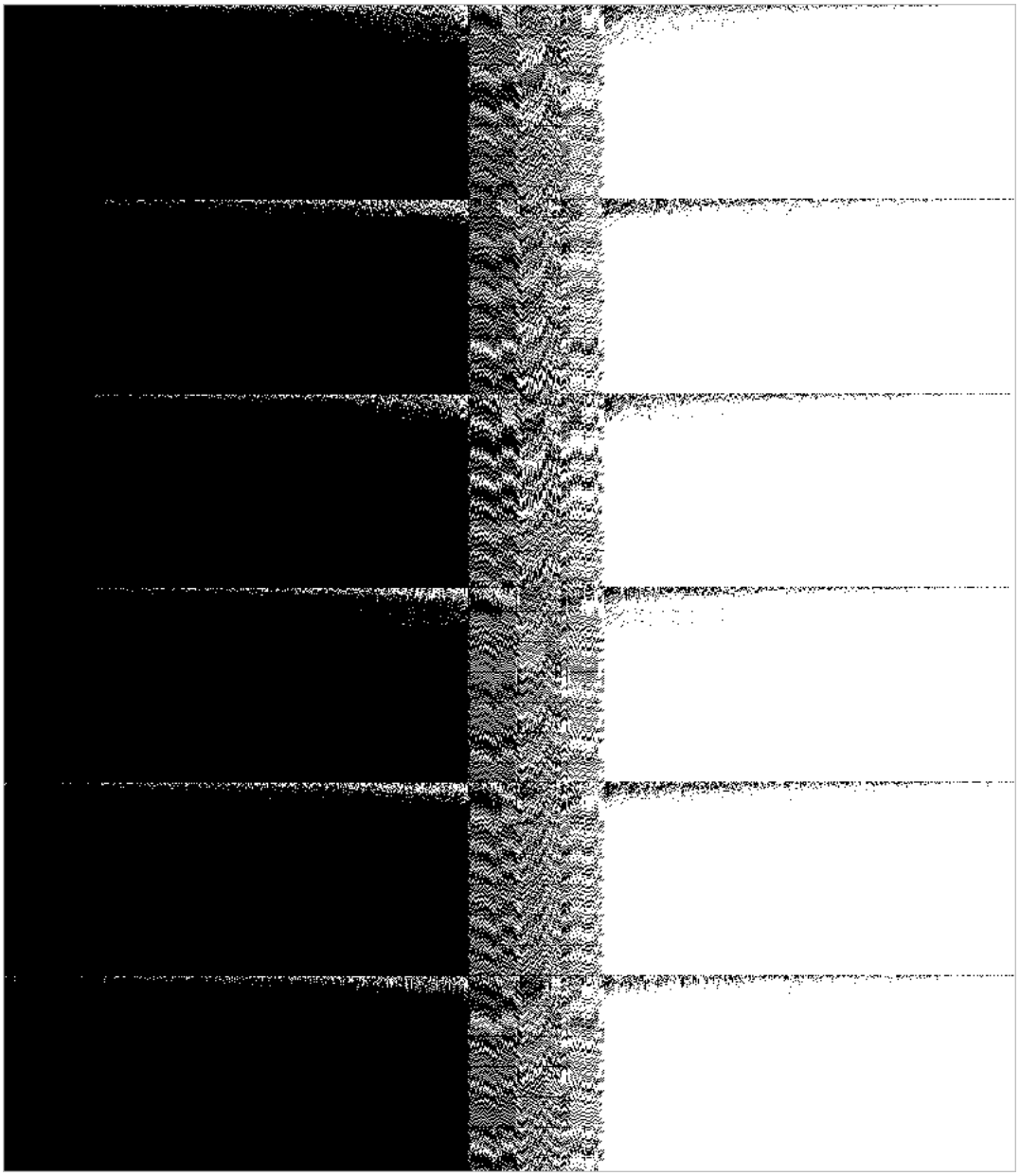

Figure 4.9 Visualisation of the time evolution of a further typical $K = 2$ RBN, with $N = 800$.

Chapter 5

Perturbing RBN state

5.1 Background

Here we visualise the stability of $K = 2$ networks to perturbations of their state.

Kauffman [3] defines a *minimal perturbation* to the state of an RBN as flipping the state of a single node at one timestep. Flipping the state of node i at time t is equivalent to changing its update rule at time $t - 1$ to be $c_{i,t} = \neg\phi_i(\chi_{i,t-1})$. Such a perturbation leaves the underlying dynamics, and hence the attractor basin structure, the same, it merely moves the current state to a different position in the state space, from where it continues to evolve under the original dynamics: it is a transient perturbation to the state.

Kauffman [3] describes the *stability* of RBN attractors to minimal perturbations: if the system is on an attractor and suffers a minimal perturbation, does it return to the same attractor, or move to a different one? Is the system *homeostatic*? (Homeostasis is the tendency to maintain a constant state, and to restore its state if perturbed.)

Kauffman [4] describes the *reachability* of other attractors after a minimal perturbation: if the system moves to a different attractor, is it likely to move to any other attractor, or just a subset of them? If the current attractor is considered the analogue of "cell type", how many other types can it *differentiate* into under minimal perturbation?

Kauffman's results are summarised in table 2.1, which picks out the $K = 2$ networks as having interesting behaviour under minimal perturbation (high stability so a perturbation usually has no effect; low reachability so when a perturbation moves the system to another attractor, it moves it to one of only a small subset of possible attractors).

5.2 Slow perturbations

Visualisations of the effect of minimal perturbations are shown in figures 5.1–5.4, for perturbations of cycling nodes, and of frozen core nodes. (See appendix B.3 for Matlab source code.) The perturbation rate is slower than the typical transient timescale, allowing the system to settle to an attractor before the next perturbation is applied.

These visualisations demonstrate that $K = 2$ RBNs are remarkably stable to minimal perturbations. They also suggest further possible properties: (a) a perturbation to a frozen core node is more likely to preserve the attractor than a perturbation to a cycling node; (b) a perturbation to a frozen core node tends to have longer transient behaviour than a perturbation to a cycling node; (c) a perturbation to a cycling node tends not to cause transients in the frozen core.

5.3 Fast perturbations

Visualisations of the effect of minimal perturbations are shown in figures 5.5 and 5.6, for perturbations of cycling nodes, and of frozen core nodes. The perturbation rate here is fast enough that the system has not necessarily reached an attractor before the next perturbation is applied, but may still be in a transient state.

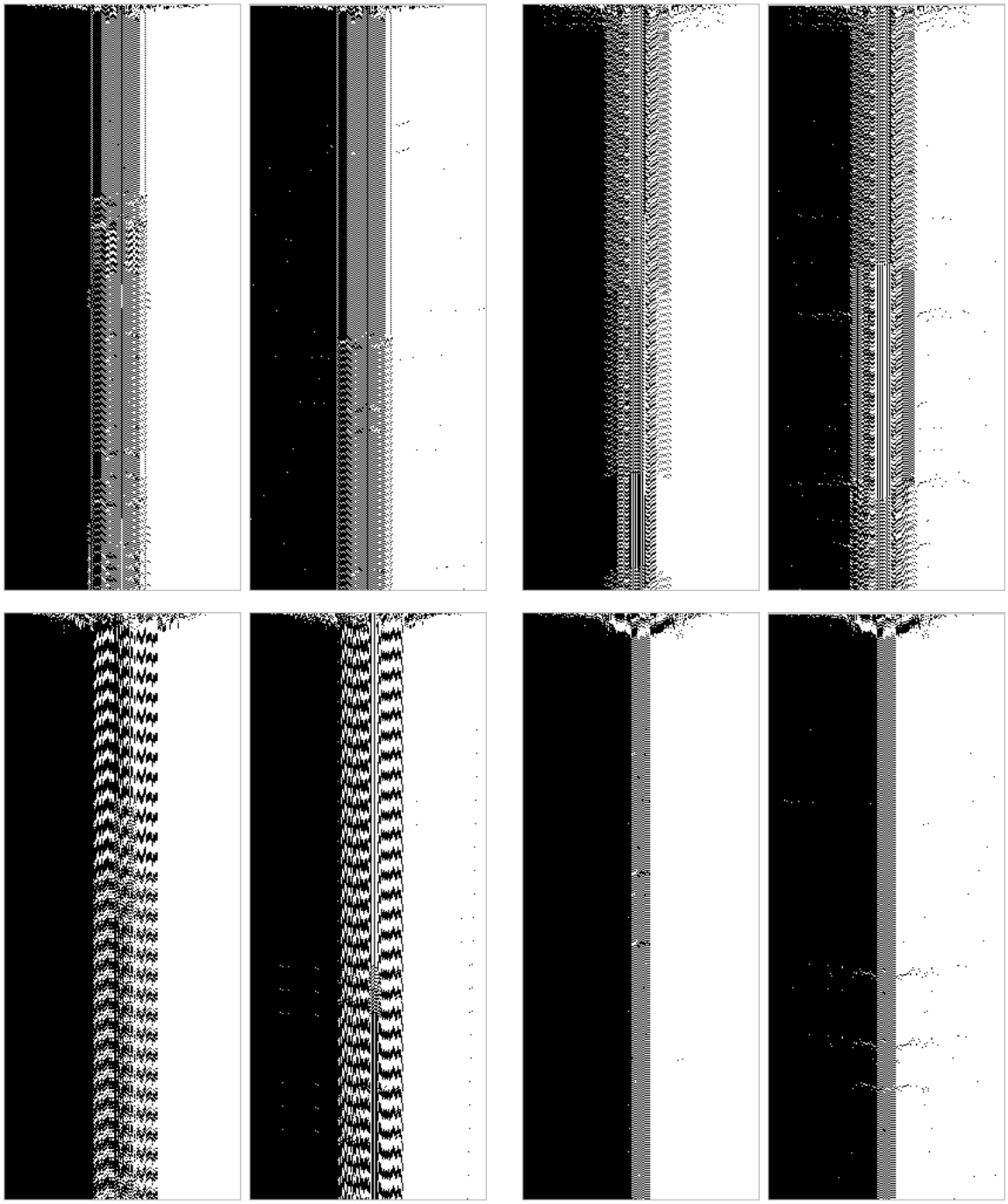

Figure 5.1 Visualisation of the time evolution of four typical $K = 2$ RBNs with $N = 200$ (two runs of each) undergoing minimal perturbation. The nodes are sorted on 4 runs of 100 timesteps. Then a single run of 500 timesteps, and random initial condition, is shown. After 100 timesteps, a node is flipped once every 20 timesteps. For the left run of each pair, a node is flipped near the centre; for the right run, a node flipped in the frozen core.

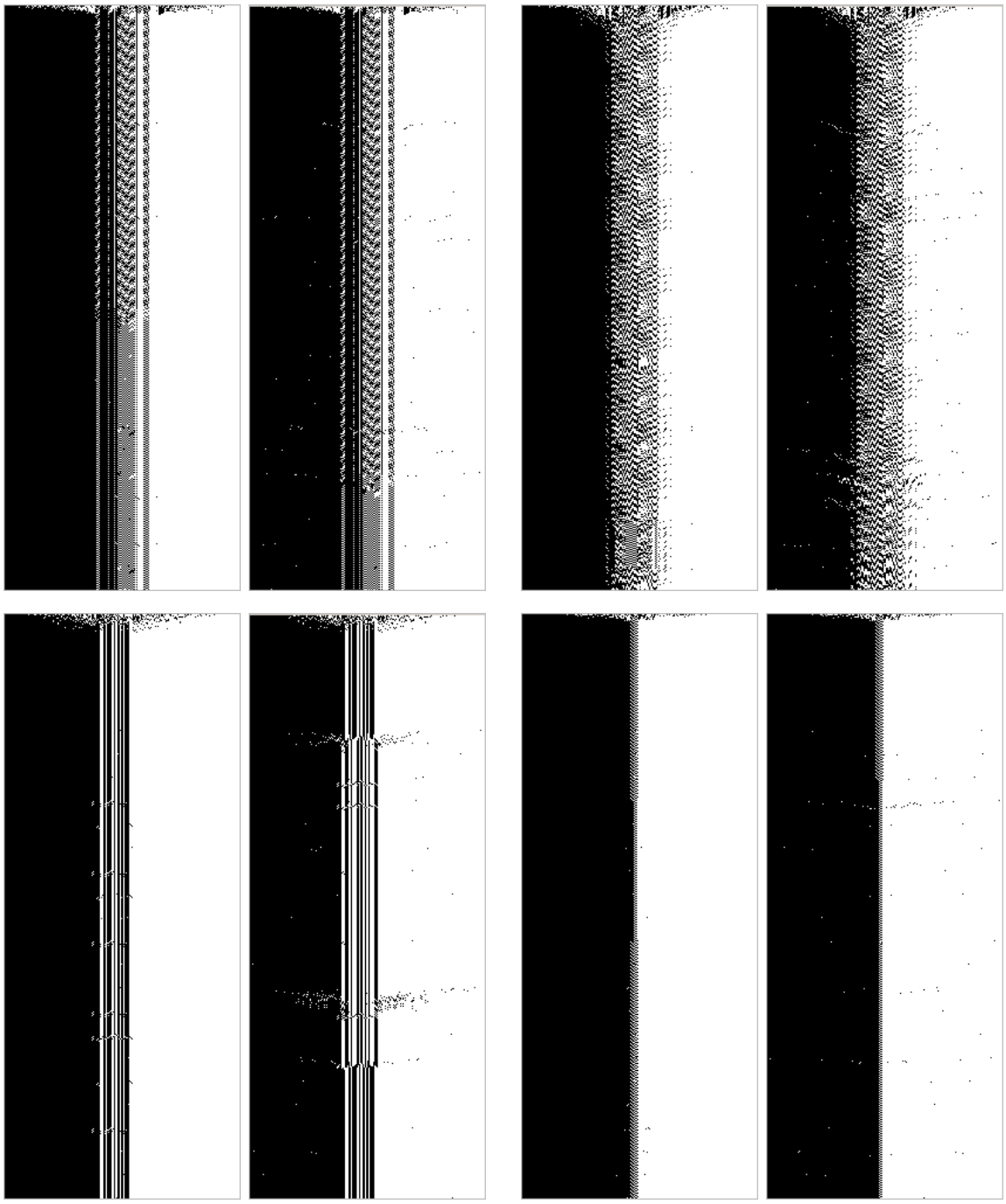

Figure 5.2 Visualisation of the time evolution of four more typical $K = 2$ RBNs with $N = 200$ (two runs of each) undergoing minimal perturbation (see figure 5.1 for details).

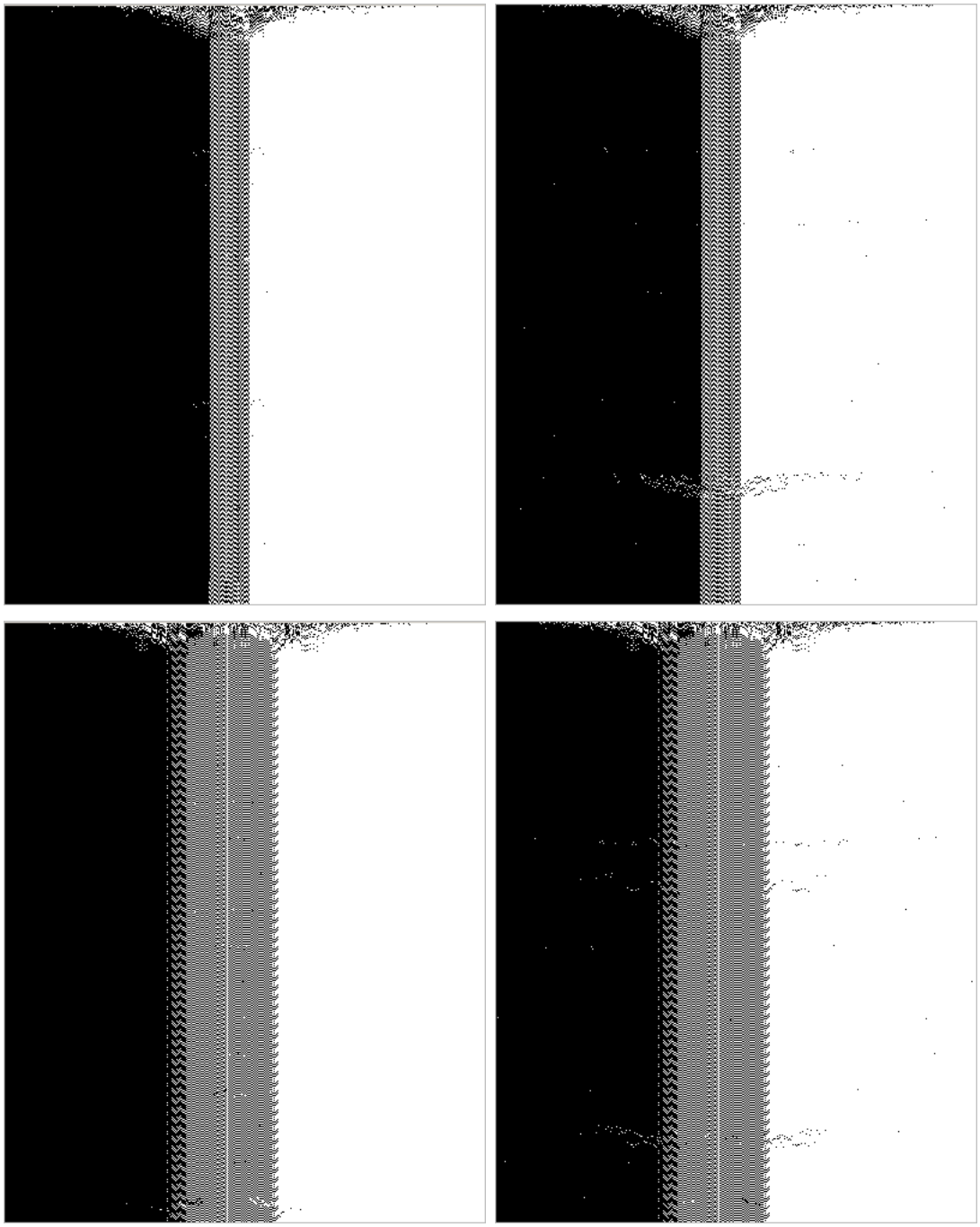

Figure 5.3 Visualisation of the time evolution of two typical $K = 2$ RBNs with $N = 400$ (two runs of each) undergoing minimal perturbation. The nodes are sorted on 4 runs of 100 timesteps. Then a single run of 500 timesteps, and random initial condition, is shown. After 100 timesteps, a node is flipped once every 30 timesteps. For the left run of each pair, a node is flipped near the centre; for the right run, a node flipped in the frozen core.

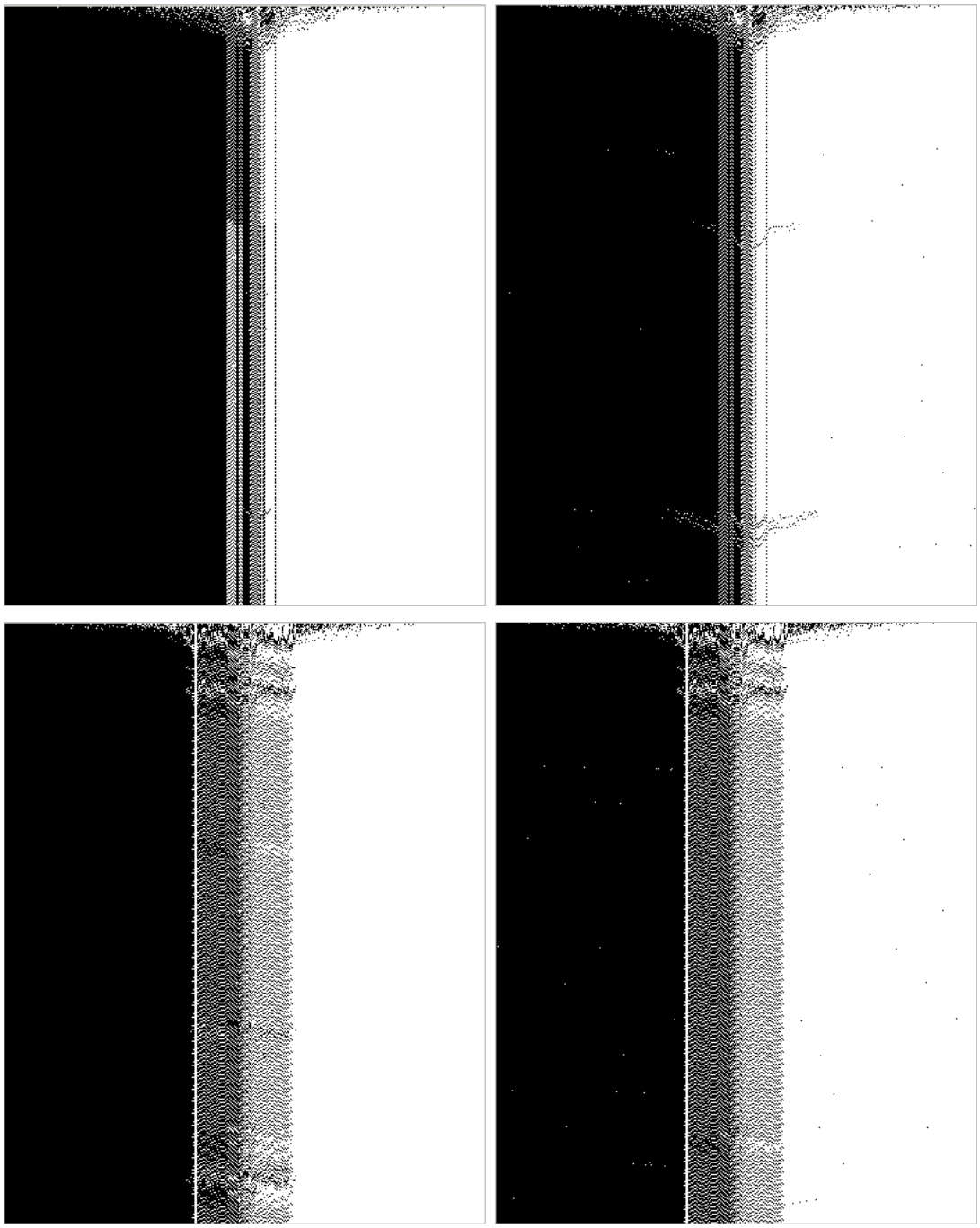

Figure 5.4 Visualisation of the time evolution of two further typical $K = 2$ RBNs with $N = 400$ (two runs of each) undergoing minimal perturbation (see figure 5.3 for details).

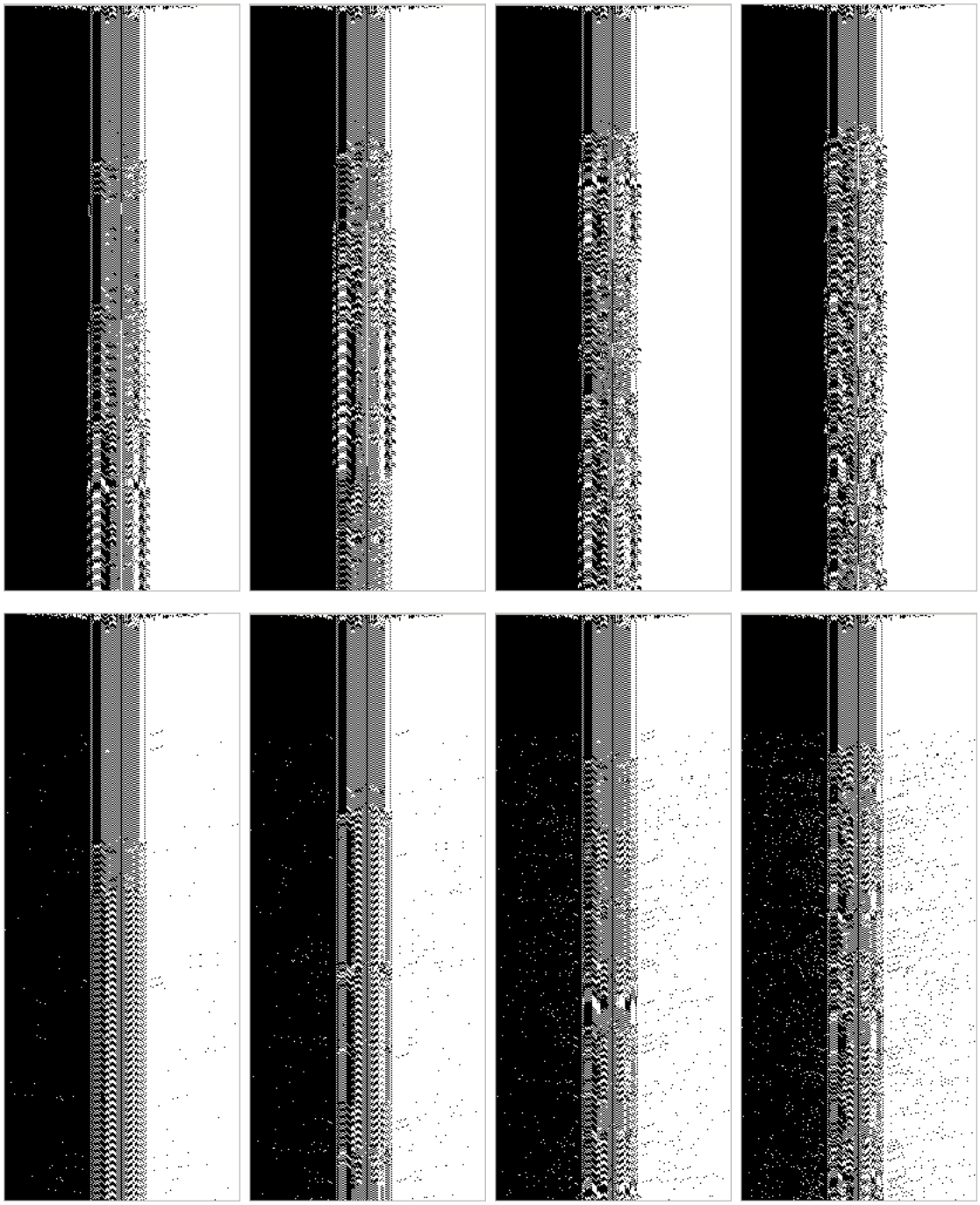

Figure 5.5 Visualisation of the time evolution of a typical $K = 2$ RBN with $N = 200$ (two runs of each) undergoing minimal (fast) perturbation. The nodes are sorted on 4 runs of 100 timesteps, Then a single run of 500 timesteps, and random initial condition, is shown. After 100 timesteps, a node is flipped once every 10,5,2,1 timesteps. For the top run of each pair, a node is flipped near the centre; for the bottom run, a node flipped in the frozen core.

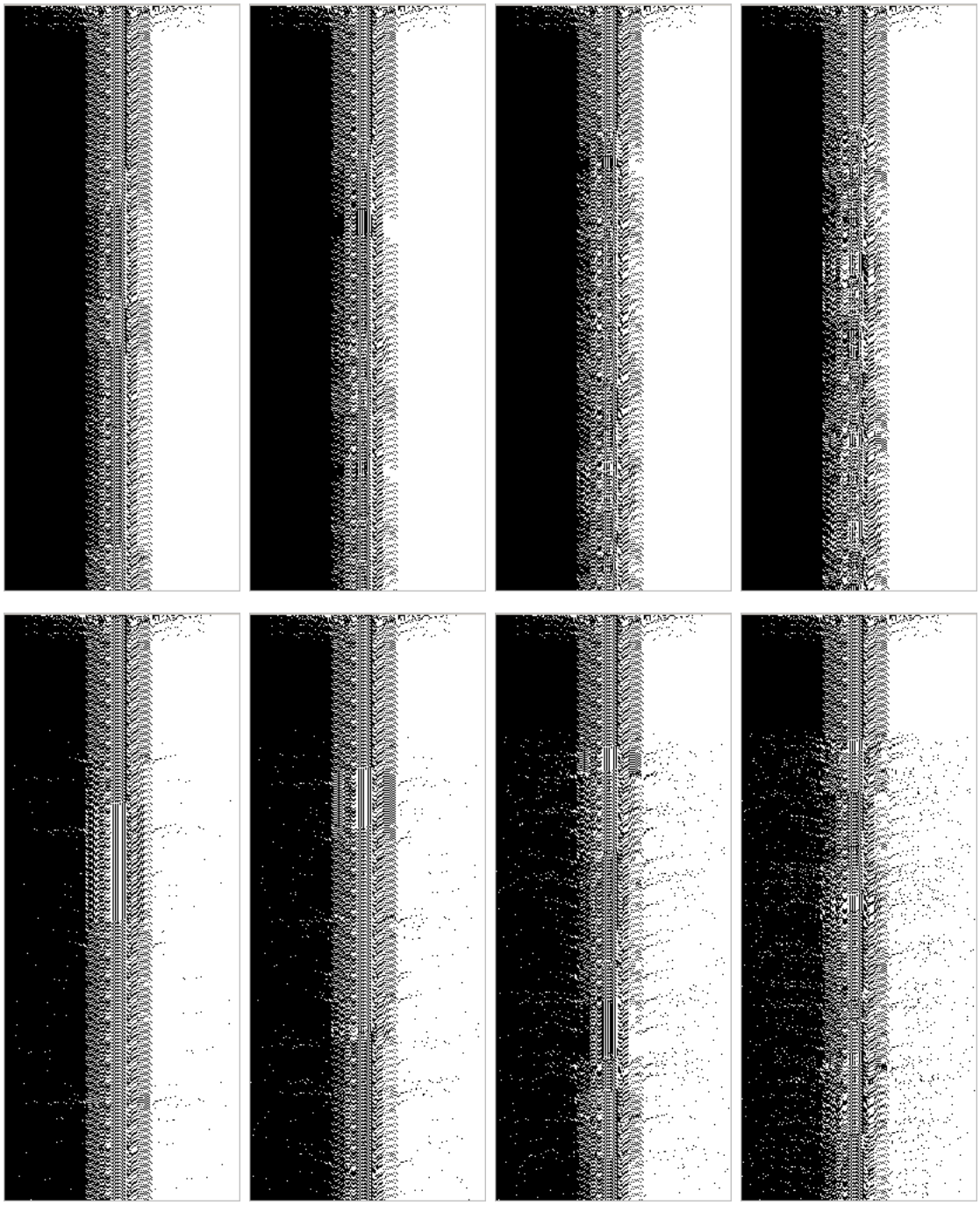

Figure 5.6 Visualisation of the time evolution of a further typical $K = 2$ RBN with $N = 200$ (two runs of each) undergoing minimal (fast) perturbation (see figure 5.5 for details).

Chapter 6

Mutating RBN structure

6.1 Background

Here we visualise the stability of $K = 2$ networks to changes of their structure.

Kauffman [3] defines a *structural perturbation* to an RBN as being a permanent mutation in the connectivity or in the boolean function. So a structural perturbation at time t_0 could change the update rule of node i at all time $t > t_0$ to be ϕ_i' or change the neighbourhood of node i at all time $t > t_0$ to be ν_i'. Since the dynamics is defined by all the ϕ_i and ν_i, such a perturbation changes the underlying dynamics, and hence the attractor basin structure: it is a permanent perturbation to the dynamics, yielding a new RBN.

Such a perturbation could have several consequences: a state previously on an attractor cycle might become a transient state; a state previously on a cycle might move to a cycle of different length, comprising different states; a state might move from an attractor with a small basin of attraction to one with a large basin; a state might move from a stable (homeostatic) attractor to an unstable attractor; and so on.

Kauffman [4] relates structural perturbation to the *mutation* of a cell; if there is only a small change to the dynamics, this represents mutation to a "similar" kind of cell.

6.2 Mutating the wiring

6.2.1 The mutation

A wiring mutation to a node changes one of its input source nodes, to some randomly chosen node. (See appendix B.4 for Matlab source code.)

6.2.2 Slow mutations

Visualisations of the effect of mutating input connections are shown in figures 6.1–6.4. The mutation rate is slower than the typical transient timescale, allowing the system to settle to an attractor before the next mutation is applied.

Mutating wiring of central active nodes tends, in the majority of cases, to "simplify" the attractor structure, adding more nodes to the frozen core. This seems reasonable: an active node must have at least one input from another active node, but, according to Drossel [1, §5.A], "the number of nodes that are nonfrozen and that receive 2 nonfrozen inputs is proportional to $N^{1/3}$". So the majority of active nodes have only one active input: if the mutation changes this to an input from a frozen node the active node will become frozen.

Mutating wiring of a frozen core node also, in most cases, has minimal effect. But it can have a dramatic effect, if a frozen node gets input from an active node and that change then propagates further (see, for example, the lower right RBN in figure 6.3).

6.2.3 Fast mutations

Visualisations of the effect of mutating input connections are shown in figures 6.5 and 6.6, for mutations of cycling nodes, and of frozen core nodes. The mutation rate here is fast enough that the system has not necessarily reached an attractor before the next mutation is applied, but may still be in a transient state.

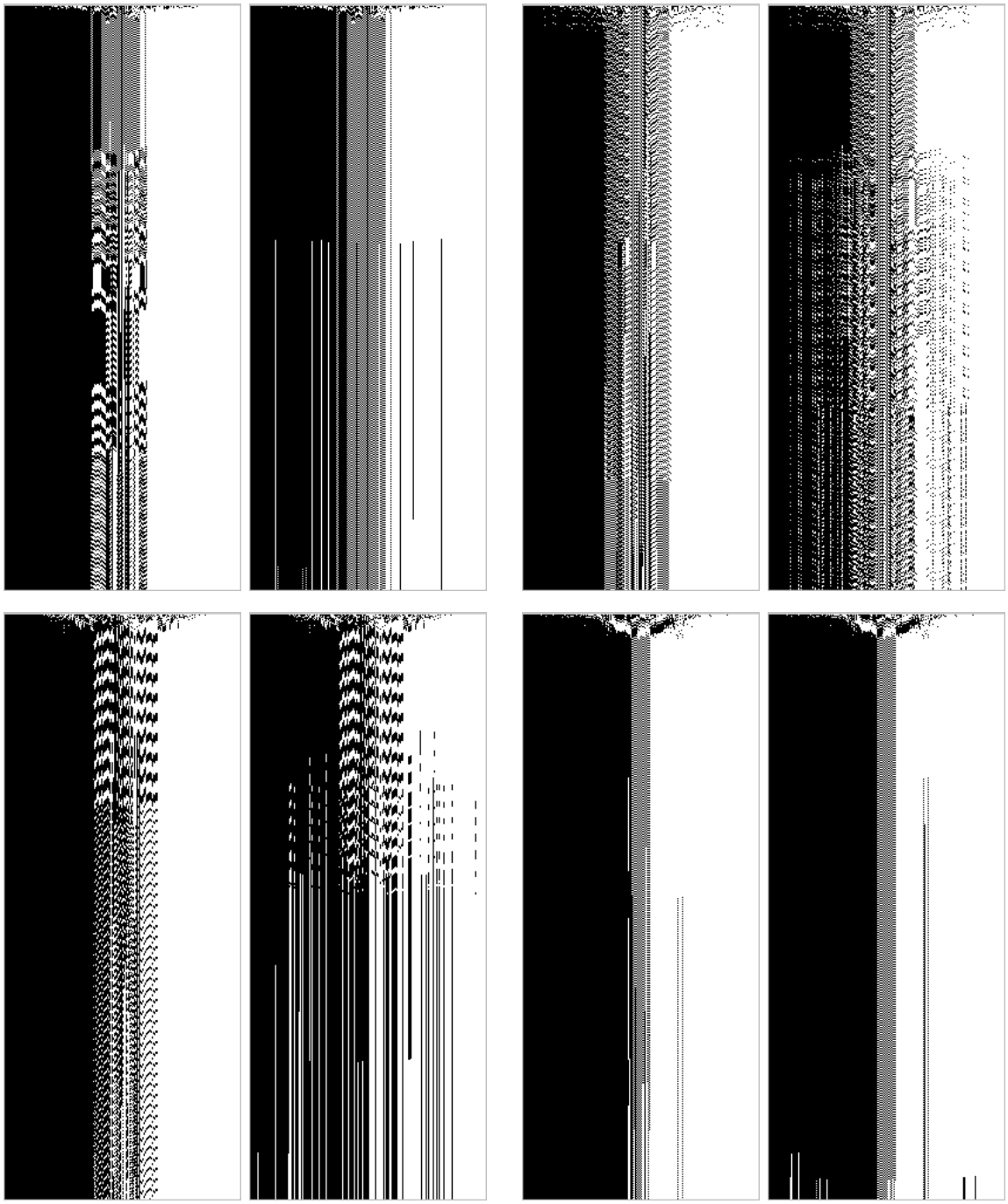

Figure 6.1 Visualisation of the time evolution of four typical $K = 2$ RBNs with $N = 200$ (two runs of each) undergoing wiring mutation. The nodes are sorted on 4 runs of 100 timesteps, Then a single run of 500 timesteps, and random initial condition, is shown. After 100 timesteps, the input of a node is mutated, once every 20 timesteps. For the left run of each pair, a node is mutated near the centre; for the right run, a node is mutated in the frozen core.

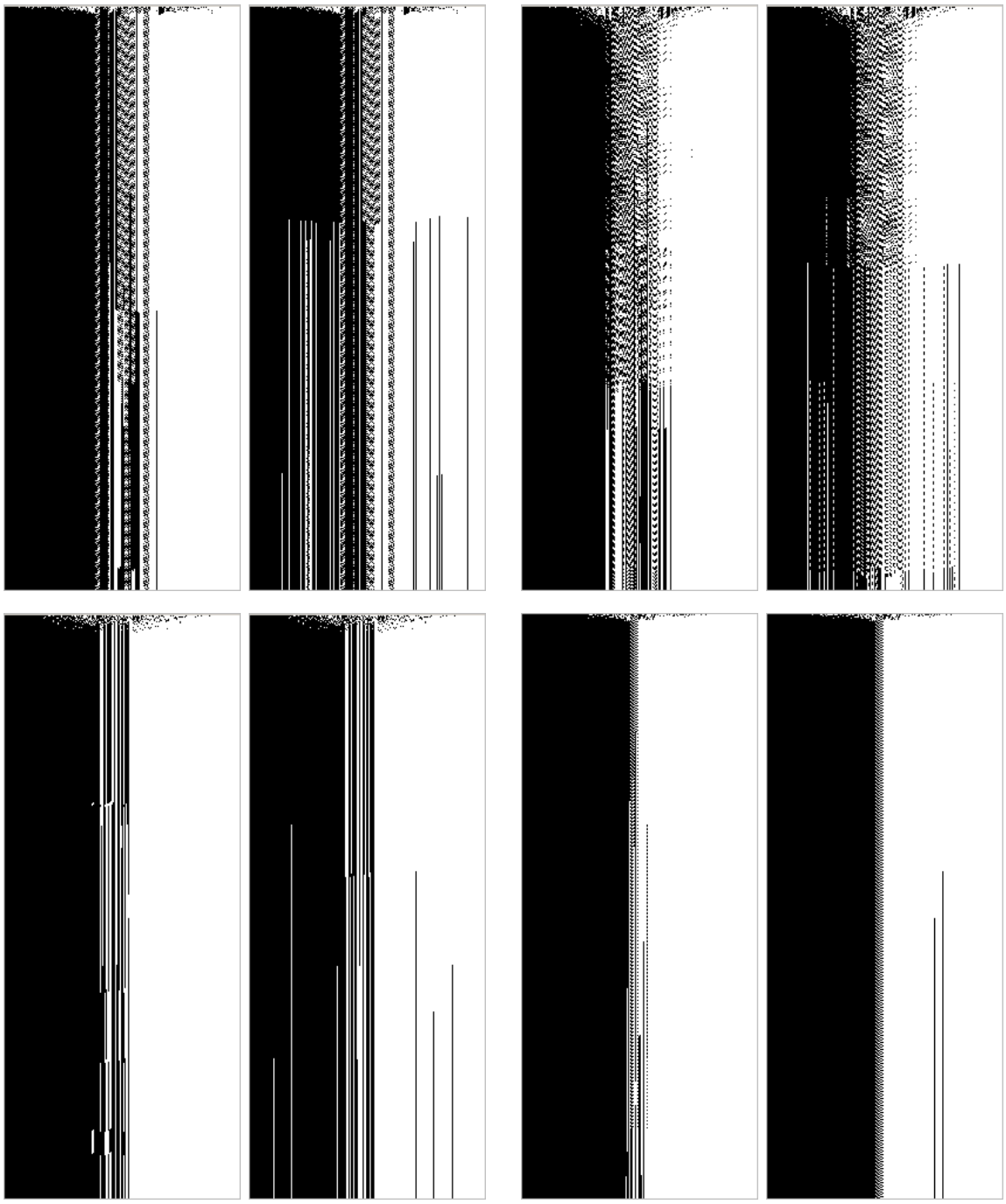

Figure 6.2 Visualisation of the time evolution of four more typical $K = 2$ RBNs with $N = 200$ (two runs of each) undergoing wiring mutation (see figure 6.1 for details).

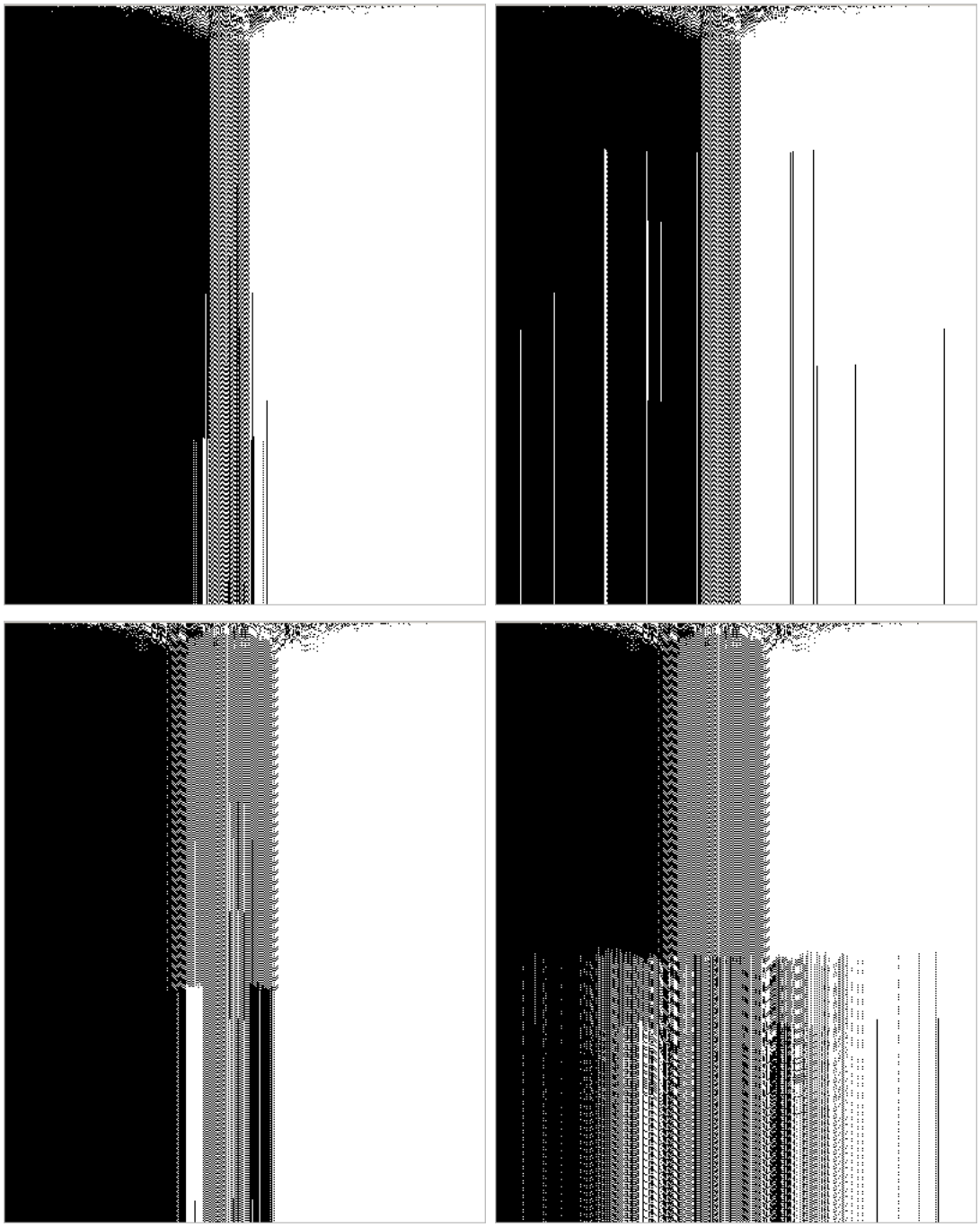

Figure 6.3 Visualisation of the time evolution of two typical $K = 2$ RBNs with $N = 400$ (two runs of each) undergoing wiring mutation. The nodes are sorted on 4 runs of 100 timesteps, Then a single run of 500 timesteps, and random initial condition, is shown. After 100 timesteps, the input of a node is mutated, once every 30 timesteps. For the left run of each pair, a node is mutated near the centre; for the right run, a node is mutated in the frozen core.

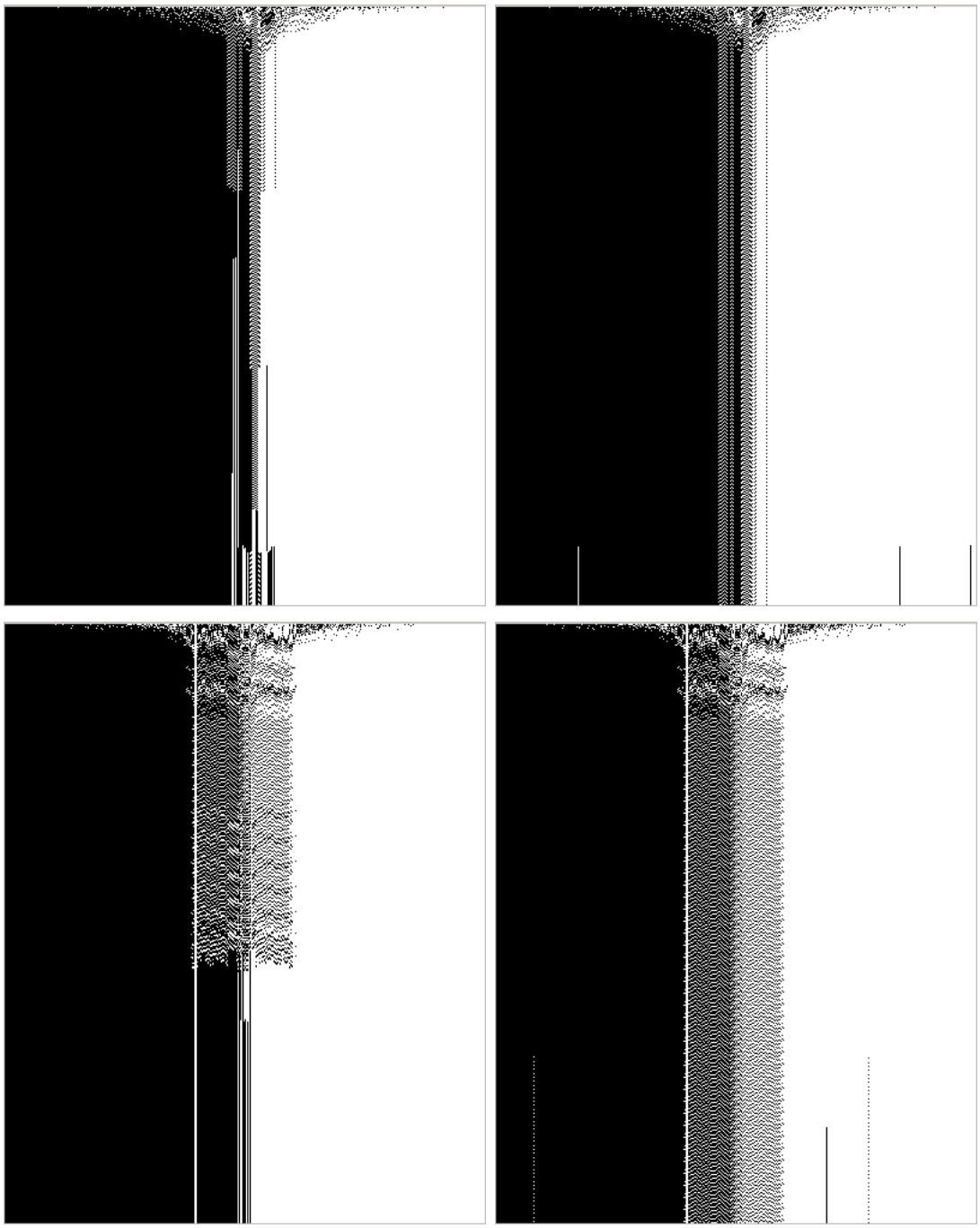

Figure 6.4 Visualisation of the time evolution of two further typical $K = 2$ RBNs with $N = 400$ (two runs of each) undergoing wiring mutation (see figure 6.3 for details).

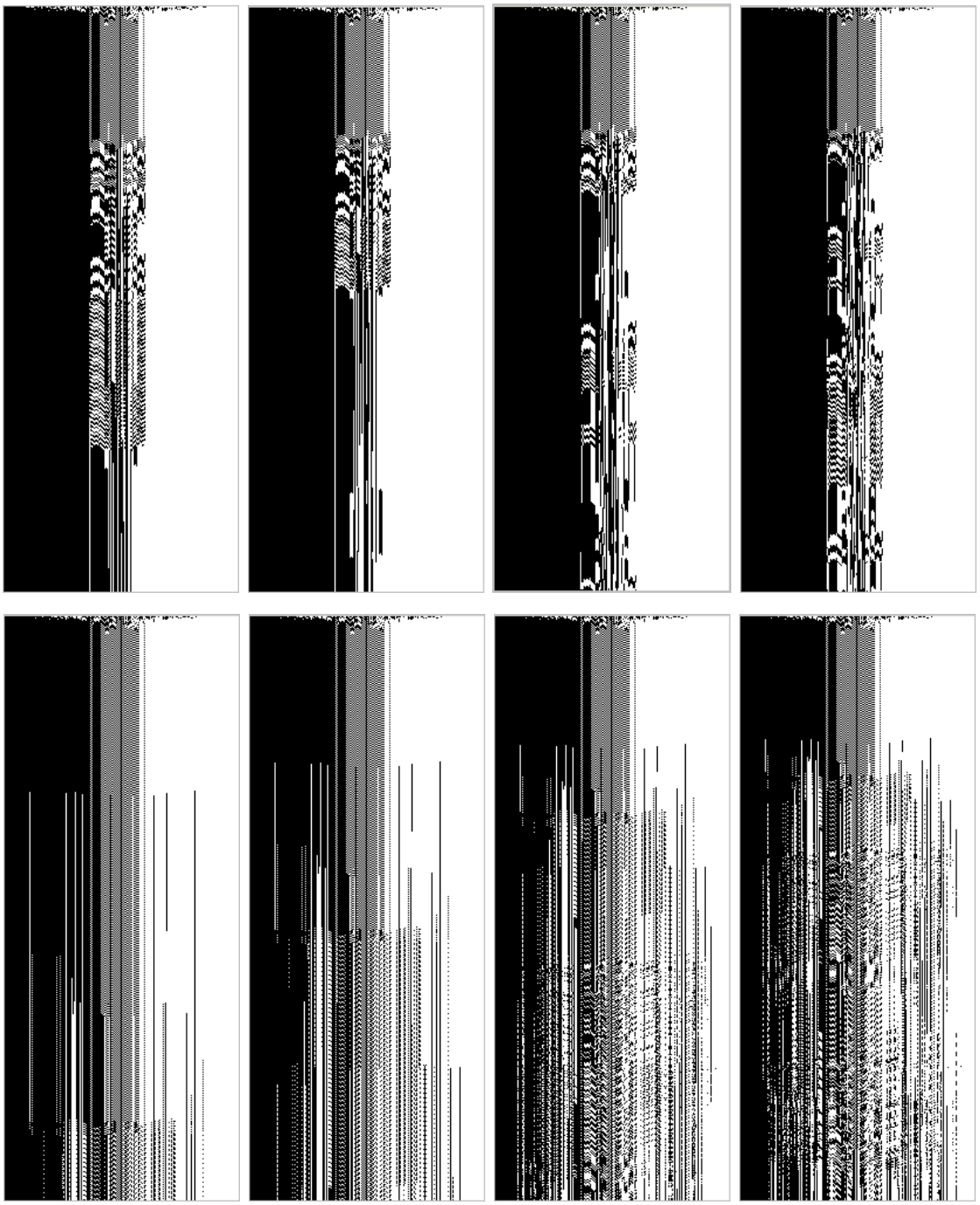

Figure 6.5 Visualisation of the time evolution of a typical $K = 2$ RBN with $N = 200$ (two runs of each) undergoing (fast) wiring mutation. The nodes are sorted on 4 runs of 100 timesteps, Then a single run of 500 timesteps, and random initial condition, is shown. After 100 timesteps, an input to a node is mutated once every 10,5,2,1 timesteps. For the top run of each pair, a node is mutated near the centre; for the bottom run, a node mutated in the frozen core.

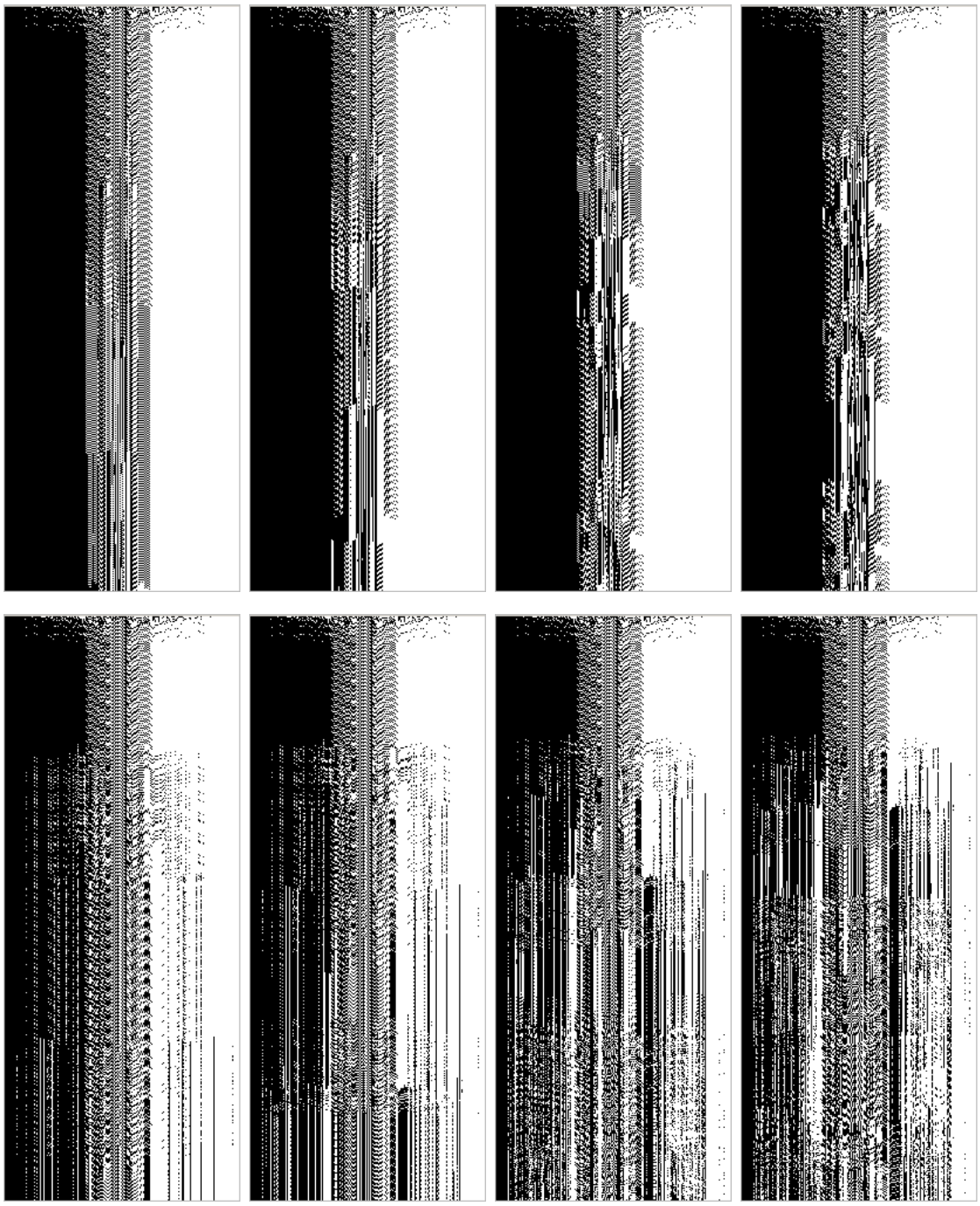

Figure 6.6 Visualisation of the time evolution of a further typical $K = 2$ RBN with $N = 200$ (two runs of each) undergoing (fast) wiring mutation (see figure 6.5 for details).

6.3 Mutating the boolean function

6.3.1 The mutation

A boolean function mutation to a node changes its boolean function to some randomly chosen function. (See appendix B.5 for Matlab source code.)

6.3.2 Slow mutations

Visualisations of the effect of mutating the boolean functions are shown in figures 6.7–6.10. The mutation rate is slower than the typical transient timescale, allowing the system to settle to an attractor before the next mutation is applied.

Changing the boolean function of an active node has a relatively small effect on the dynamics. If it mutated to a constant function, it would have the effect of freezing to node (and possible freezing downstream nodes). But it can also simply change it to a different kind of active node.

Changing the boolean function of a frozen core node, on the other hand, appears to have a larger effect. If the node was in the frozen core because it had a constant function, for example, it could become active, and potentially activate downstream nodes. In some cases, this activation process is "catastrophic", resulting in large changes to the activity, and to the dynamics of the RBN.

Visualisation of further experiments along these lines could yield interesting conjectures about the stability of these RBNs.

6.3.3 Fast mutations

Visualisations of the effect of mutating boolean functions are shown in figures 6.11 and 6.12, for mutations of cycling nodes, and of frozen core nodes. The mutation rate here is fast enough that the system has not necessarily reached an attractor before the next mutation is applied, but may still be in a transient state.

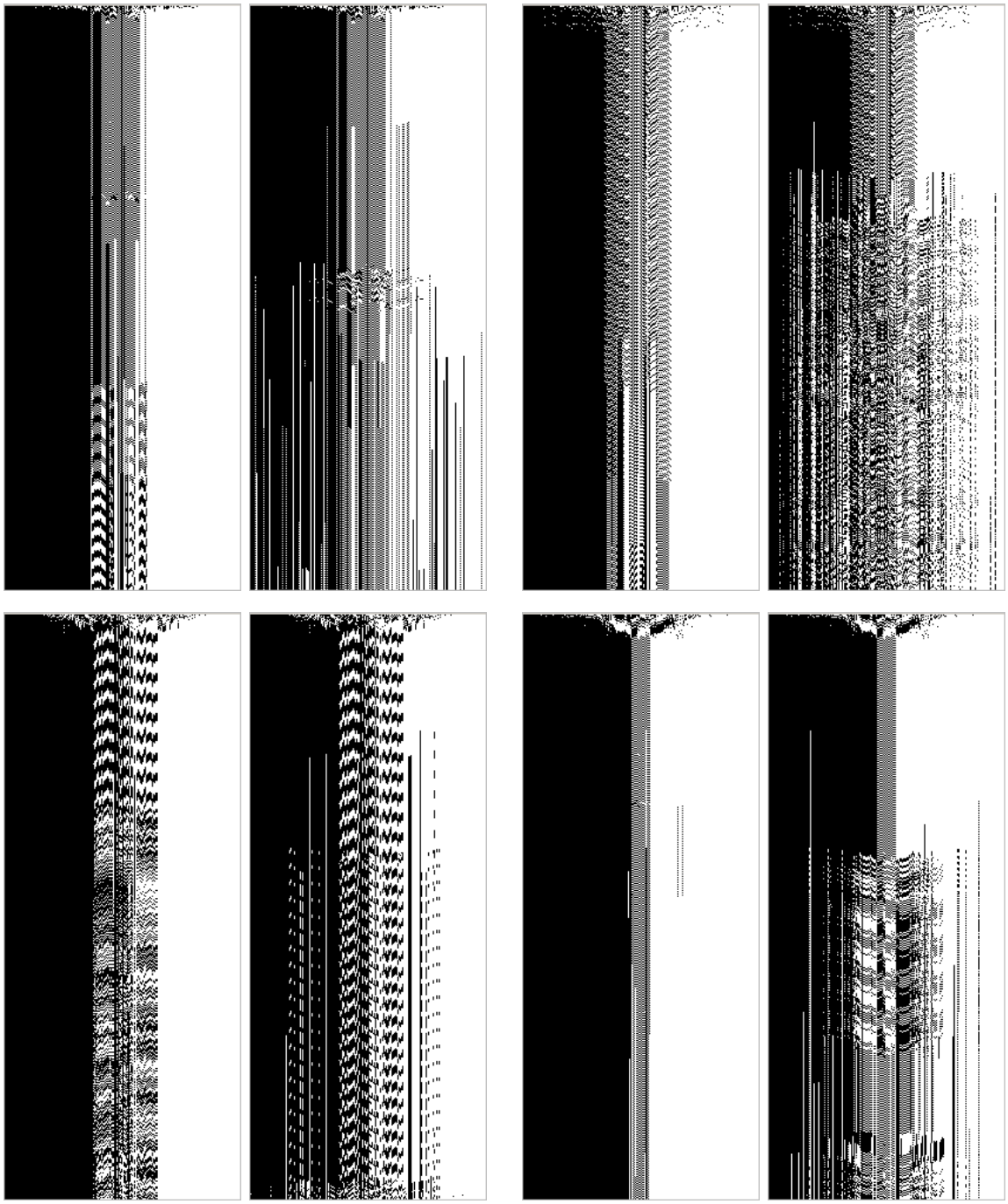

Figure 6.7 Visualisation of the time evolution of four typical $K = 2$ RBNs with $N = 200$ (two runs of each) undergoing boolean function mutation. The nodes are sorted on 4 runs of 100 timesteps, Then a single run of 500 timesteps, and random initial condition, is shown. After 100 timesteps, the boolean function of a node is mutated, once every 20 timesteps. For the left run of each pair, a node is mutated near the centre; for the right run, a node is mutated in the frozen core.

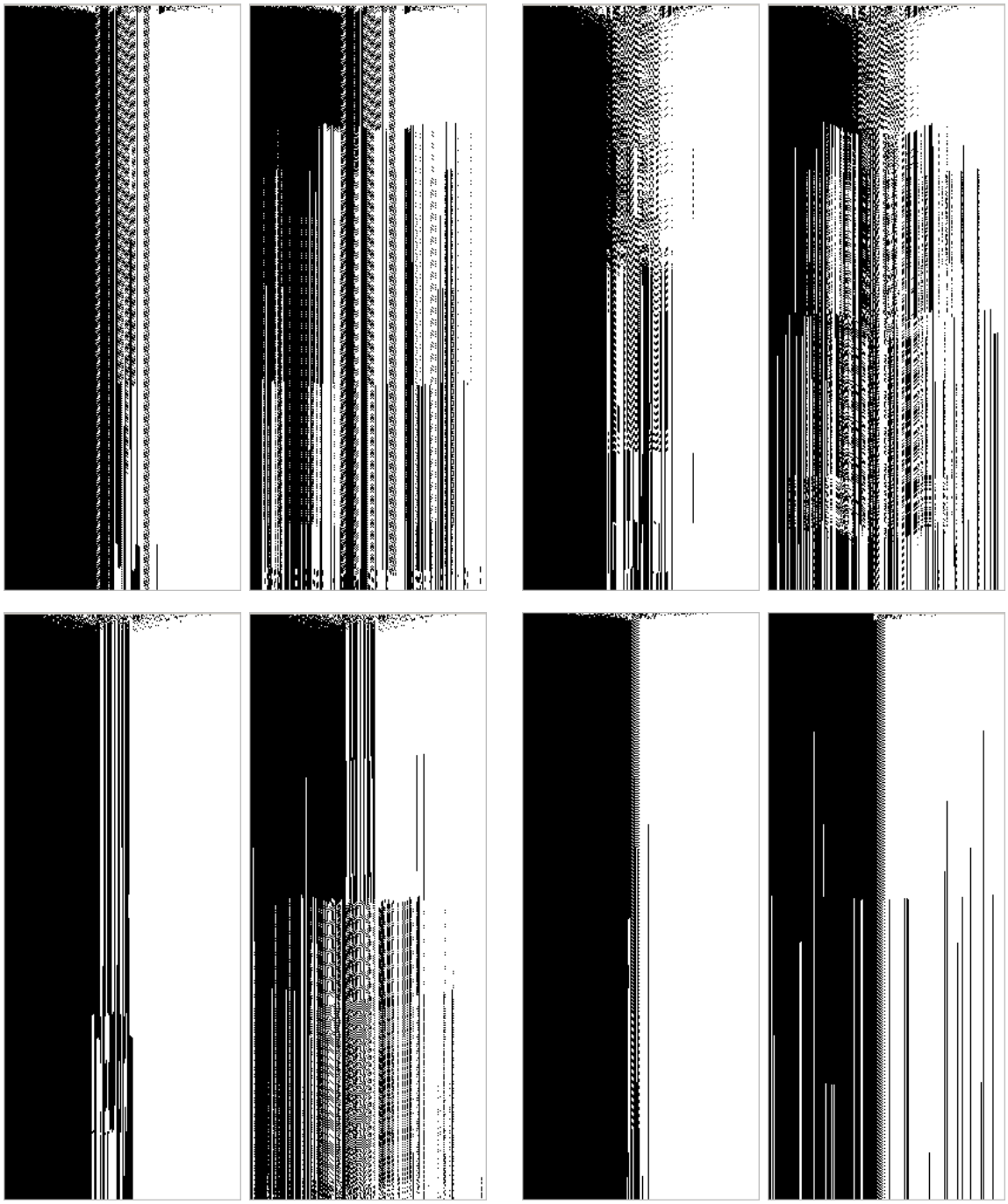

Figure 6.8 Visualisation of the time evolution of four more typical $K = 2$ RBNs with $N = 200$ (two runs of each) undergoing boolean function mutation (see figure 6.7 for details).

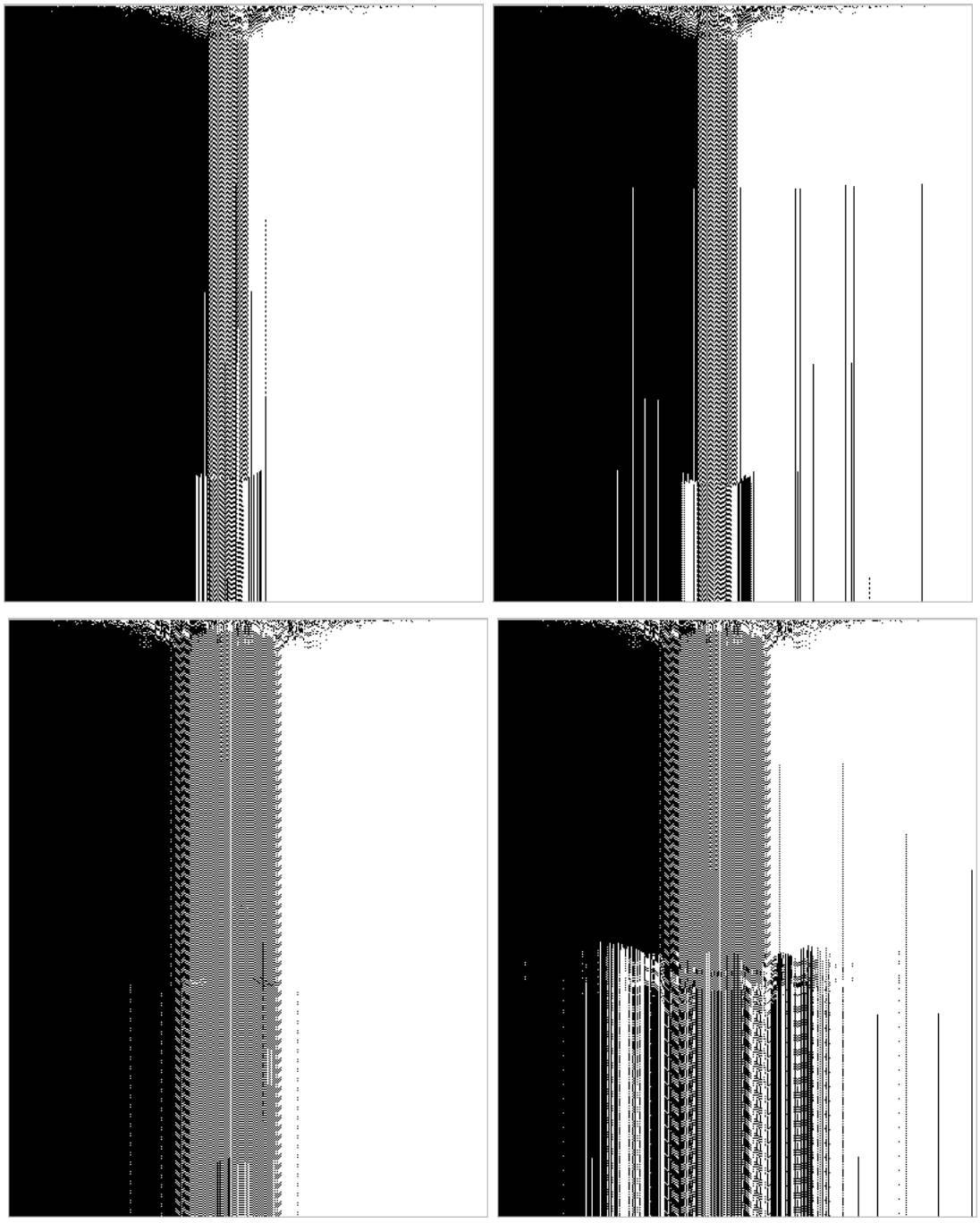

Figure 6.9 Visualisation of the time evolution of two typical $K = 2$ RBNs with $N = 400$ (two runs of each) undergoing boolean function mutation. The nodes are sorted on 4 runs of 100 timesteps, Then a single run of 500 timesteps, and random initial condition, is shown. After 100 timesteps, the boolean function of a node is mutated, once every 30 timesteps. For the left run of each pair, a node is mutated near the centre; for the right run, a node is mutated in the frozen core.

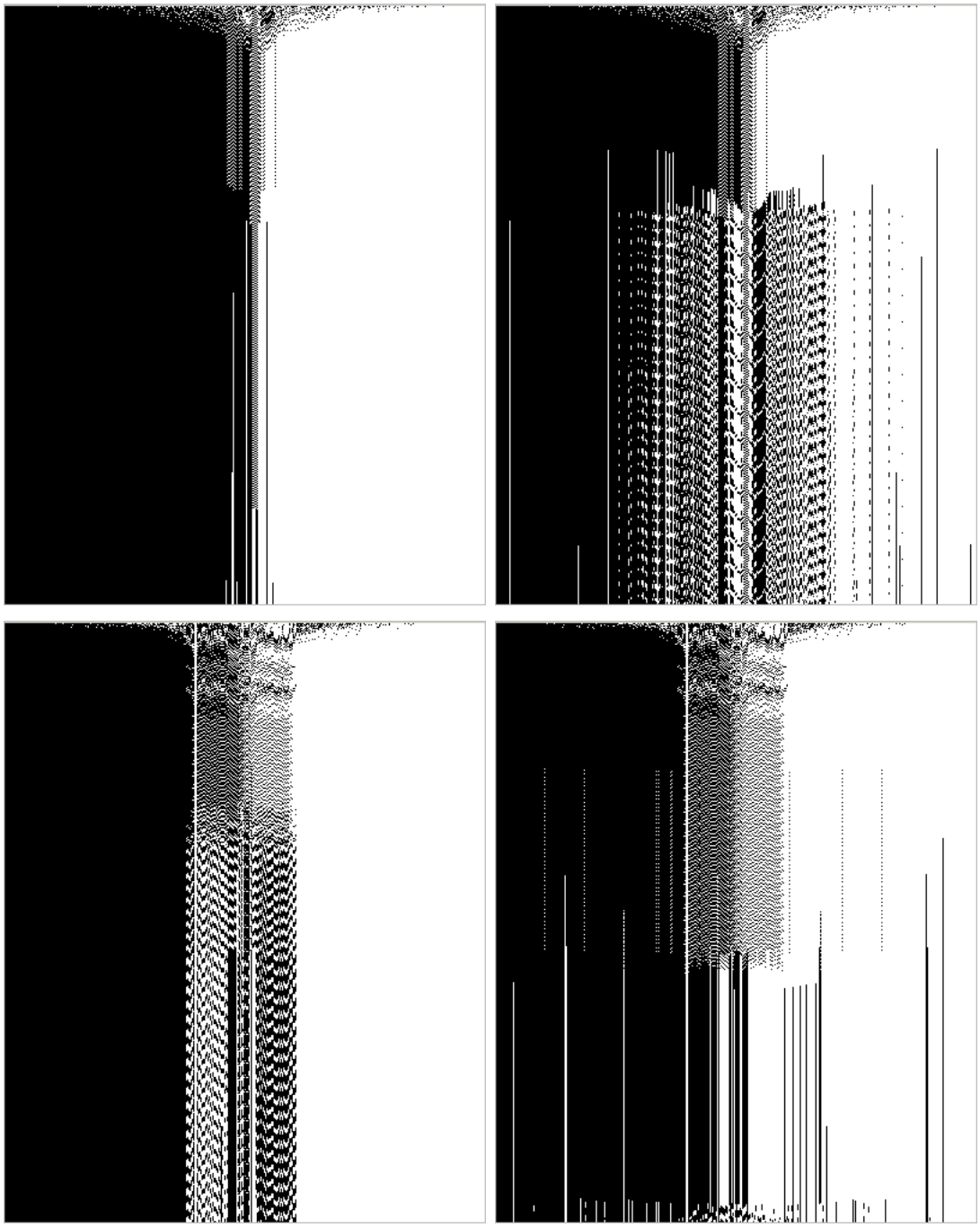

Figure 6.10 Visualisation of the time evolution of two more typical $K = 2$ RBNs with $N = 400$ (two runs of each) undergoing boolean function mutation (see figure 6.9 for details).

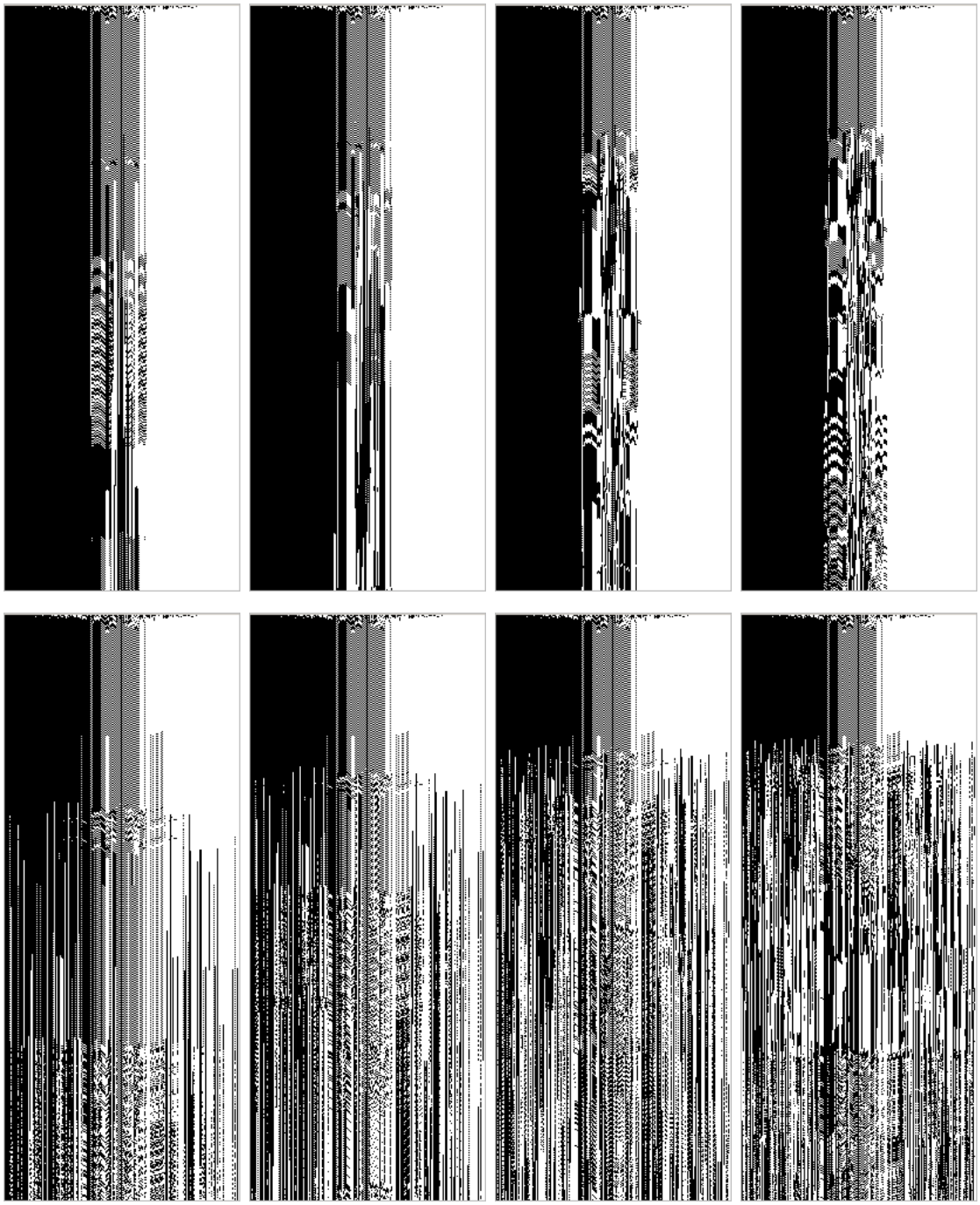

Figure 6.11 Visualisation of the time evolution of a typical $K = 2$ RBN with $N = 200$ (two runs of each) undergoing (fast) boolean function mutation. The nodes are sorted on 4 runs of 100 timesteps, Then a single run of 500 timesteps, and random initial condition, is shown. After 100 timesteps, the boolean function of a node is mutated once every 10,5,2,1 timesteps. For the top run of each pair, a node is mutated near the centre; for the bottom run, a node mutated in the frozen core.

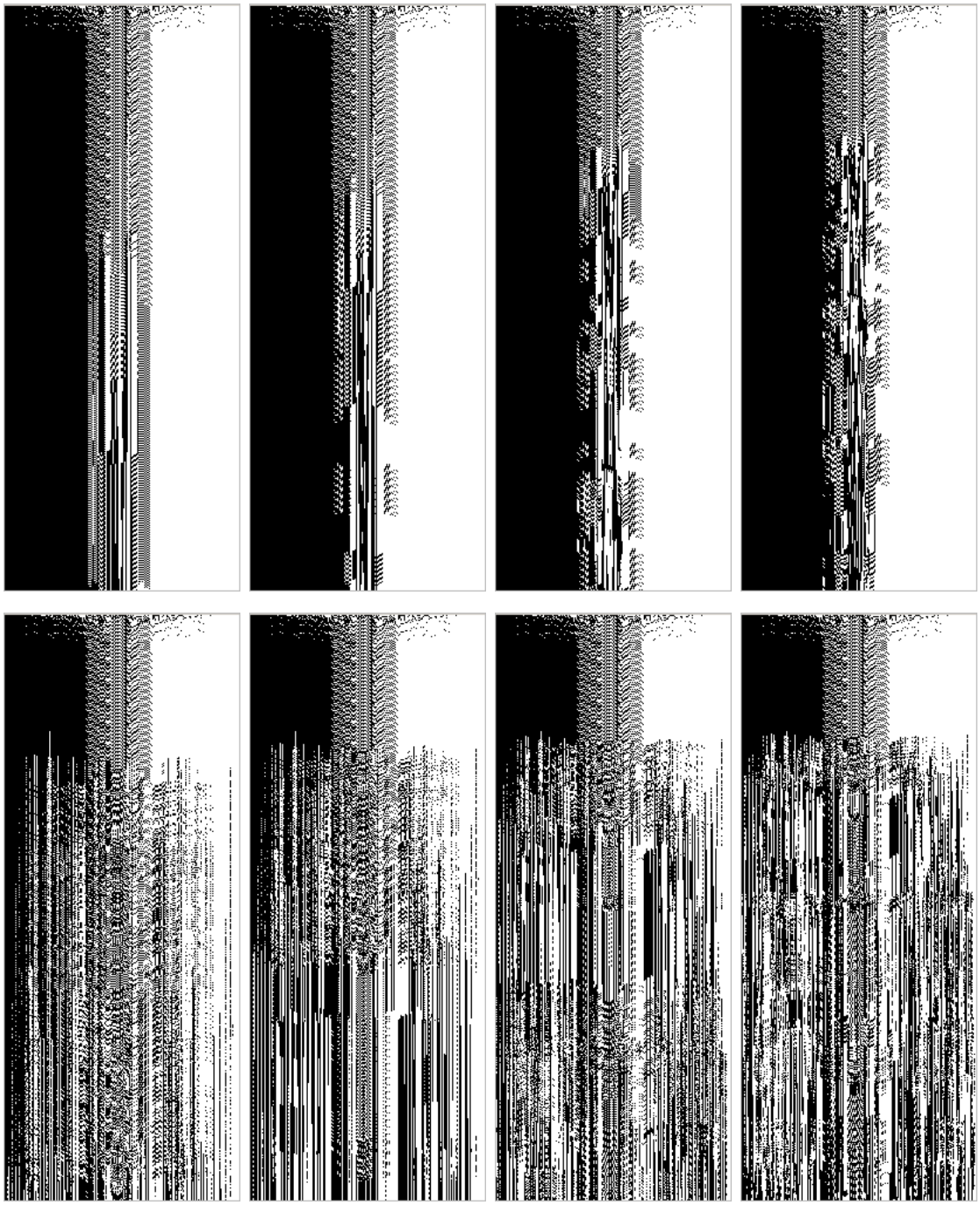

Figure 6.12 Visualisation of the time evolution of a further typical $K = 2$ RBN with $N = 200$ (two runs of each) undergoing (fast) boolean function mutation (see figure 6.11 for details).

Chapter 7

Canalisation

7.1 Introduction

Here we visualise the effect of canalising functions on the time behaviour of $K > 2$ networks. $K > 2$ networks do not show the same ordered attractor structure as $K = 2$ networks: see figure 7.1.

Kauffman [4, p.203] defines a canalising function as "any Boolean function having the property that it has at least one input having at least one value (1 or 0) which suffices to guarantee that the regulated element assumes a specific value (1 or 0)." So, for example, AND is canalising, because if one input is 0, the output is 0 regardless of the other input value; XOR is not canalising, because both inputs are always needed to determine the output. Any function that ignores all, or all but one, of its inputs is canalising.

Drossel [1, §2.B] categorises canalising functions further, into weak (where one input has one value that determines the output, so the other input values are irrelevant when the canalising input has its canalising value, for example, AND, OR), strong (where one input completely determines the output, so the other inputs are always irrelevant), and constant.

Kauffman argues that the canalising functions are important for establishing the frozen core and ordered dynamics of $K = 2$ networks. The proportion of canalising functions decreases rapidly with increasing K (see table 7.1). Kauffman [4, p.206] states that "networks with $K > 2$ restricted to canalyzing functions ... [have] orderly dynamics in the entire network".

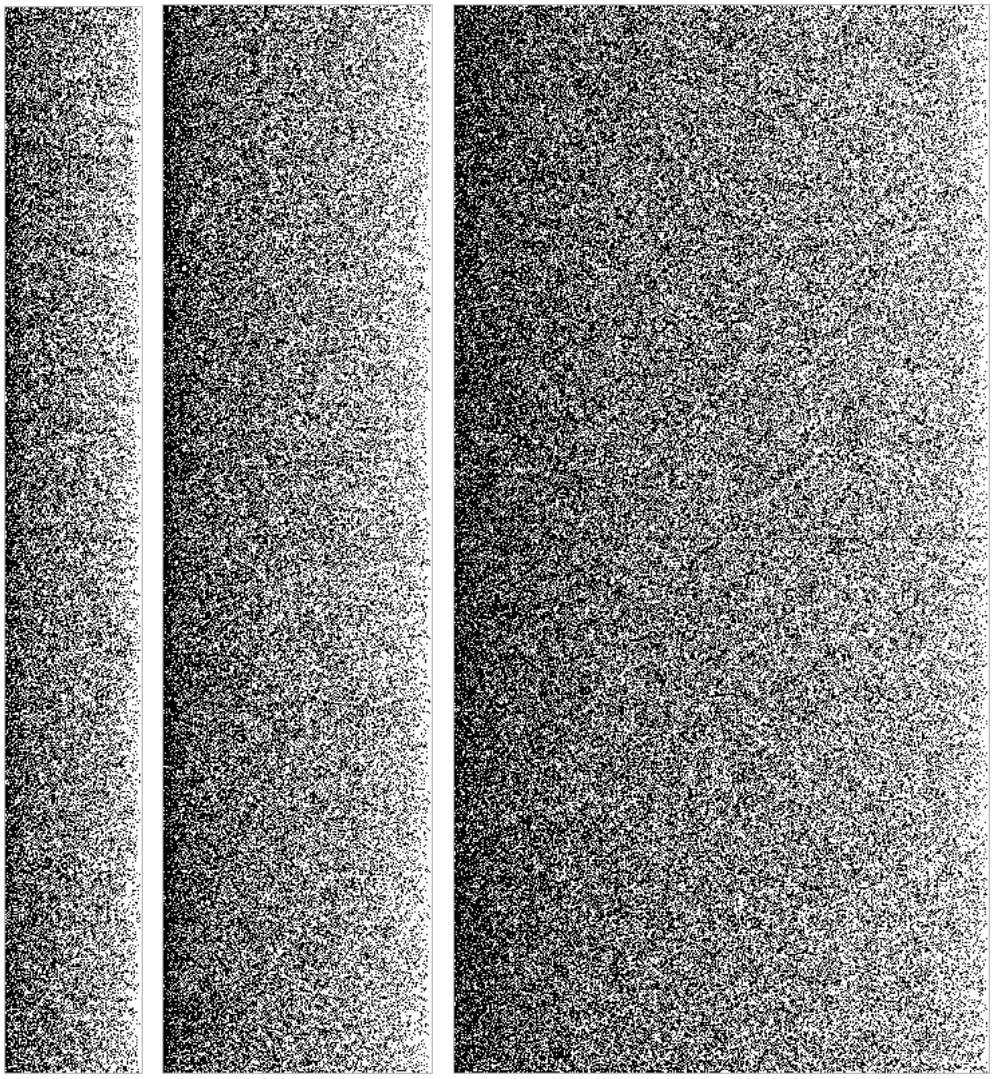

Figure 7.1 Visualisation of the time evolution of three typical $K = 3$ RBNs, with $N = 100, 200, 400$. After 400 timesteps, the nodes are reinitialised to a new random configuration, to explore another attractor. No structure is visible.

K	boolean functions	canalising functions number	proportion
2	16	14	87.50%
3	256	120	46.88%
4	65536	3514	5.36%

Table 7.1 Proportion of canalising functions

7.2 Control case: $K = 2$

Figures 7.2 and 7.3 show, as a control case, $K = 2$ networks with no bias (using all 16 boolean functions with equal probability), and all canalised (using only the 14 canalising functions) networks[1]. (See appendix B.6 for Matlab source code.)

The smaller networks ($N = 100$) show no obvious difference between the ordinary and canalised cases. The large networks ($N = 400$) appear to show a reduction in the number of active nodes, and in the attractor cycle lengths, in the canalised case.

[1] In this chapter we are always performing just a single run of a given network, so we choose to start each run from the all "on" state. This has the advantage of giving a clear visual indication of the start of the run, yet the results are qualitatively equivalent to those from random initialisation, since the on/off states are symmetrical in the definition of the RBN (that is, no special meaning is attached to either state, so the all "on" state is not a special case).

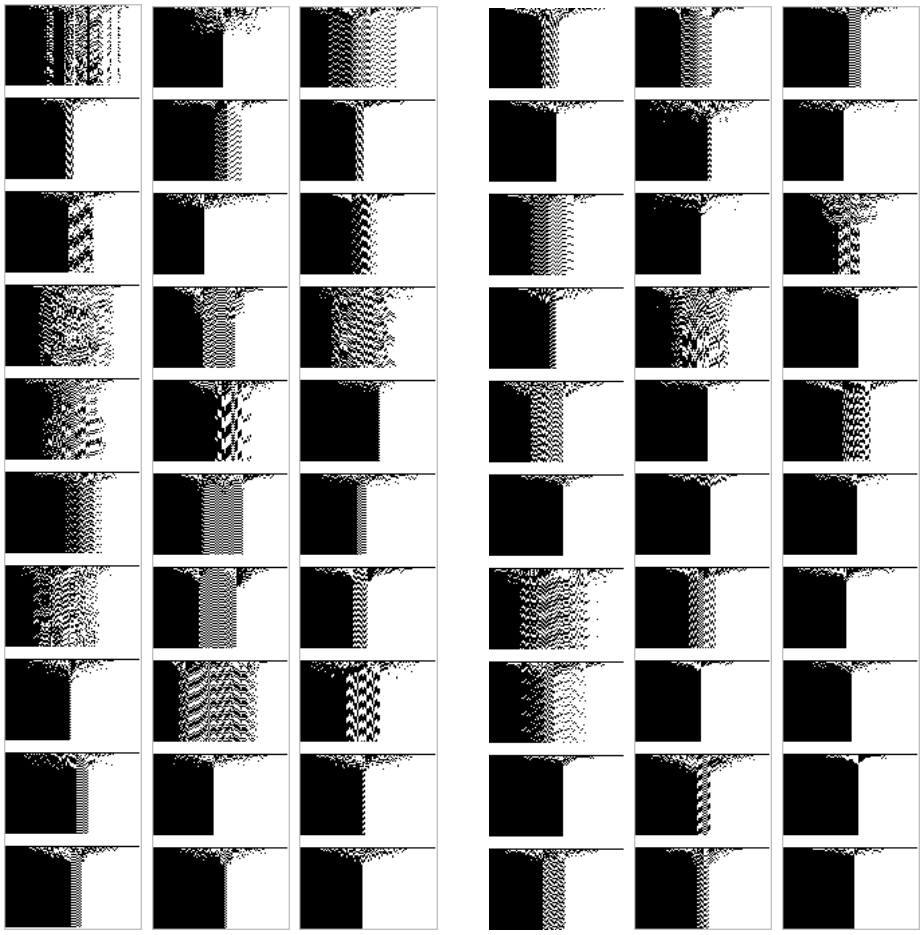

Figure 7.2 Visualisation of the time evolution of typical $K = 2$ RBNs, with $N = 100$, $t = 60$, initial condition all nodes "on"; the three columns on the left have no bias to canalised nodes; the three on the right are entirely canalised.

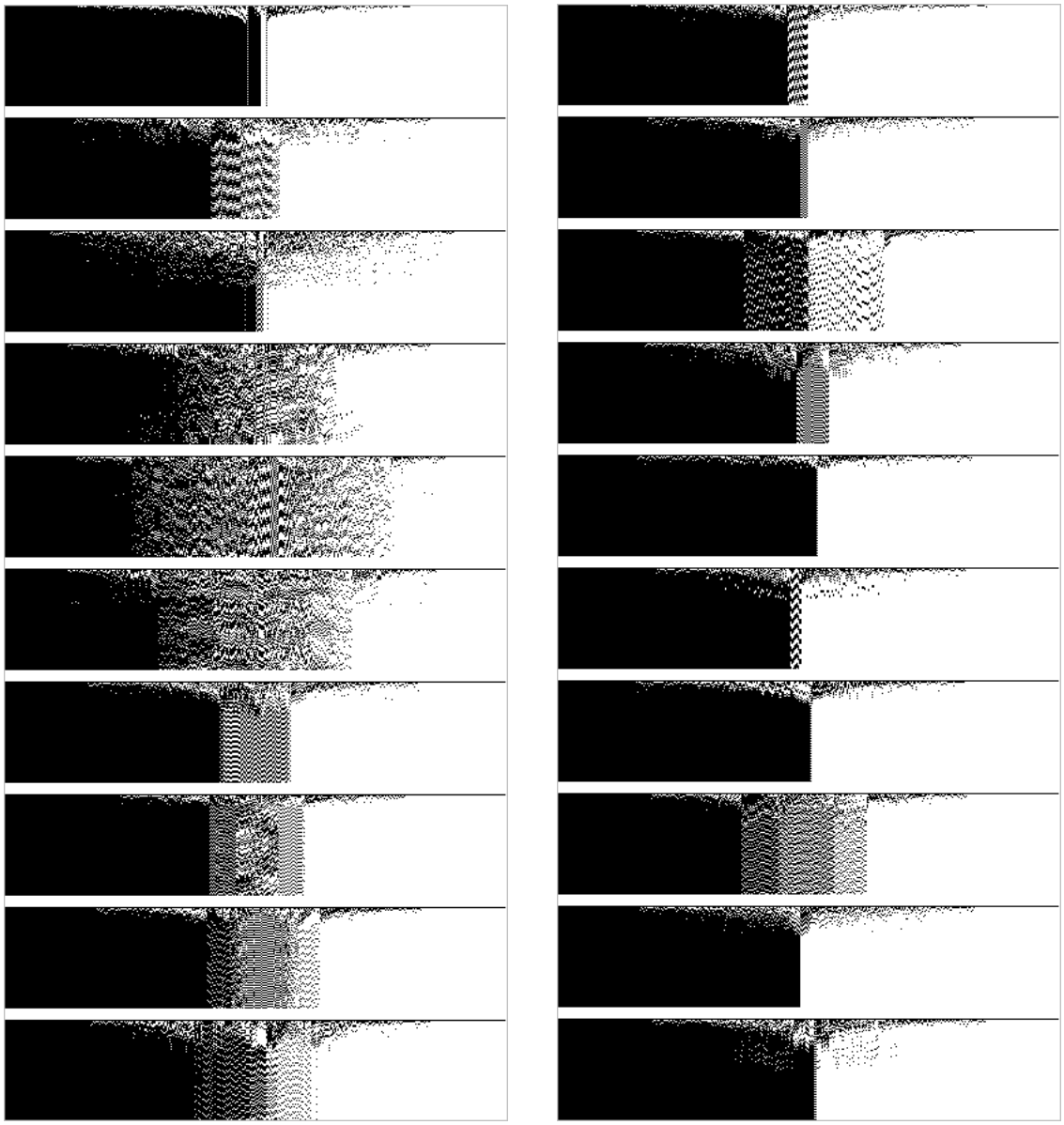

Figure 7.3 Visualisation of the time evolution of typical $K = 2$ RBNs, with $N = 400$, $t = 80$, initial condition all nodes "on"; the column on the left has no bias to canalised nodes; the column on the right is entirely canalised.

7.3 The cases $K = 3$ and $K = 4$

Visualisations of the effect of canalising functions on the time behaviour of $K = 3$ are shown in figures 7.4–7.9, and on $K = 4$ networks in figures 7.10–7.12.

Examples of canalised $N = 100$ networks are shown in figures 7.4, 7.5 ($K = 3$) and 7.10) ($K = 4$). Clearly, increasing the proportion of canalising functions does make transients and attractors shorter, and establish a more "orderly dynamics", than in the uncanalised case, although the effect is much less pronounced for $K = 4$.

For larger N (figures 7.6–7.9, 7.11, 7.12), even with all functions canalising, change in the chaotic behaviour is evident in only a minority of cases, and appears to be decreasing as N increases.

For canalised networks that take longer to reach an attractor than the typical $K = 2$ convergence time, figures 7.5, 7.7 and 7.9 show that this can be due to longer transients, longer attractors, or both. These are nevertheless still more ordered than the uncanalised $K = 3$ networks in figure 7.1.

So we can see that a canalising effect exists, resulting in more ordered dynamics, but it is not strong enough to reduce the behaviour to the $K = 2$ case. It seems that the highly ordered $K = 2$ behaviour is due to more than just the effect of canalisation.

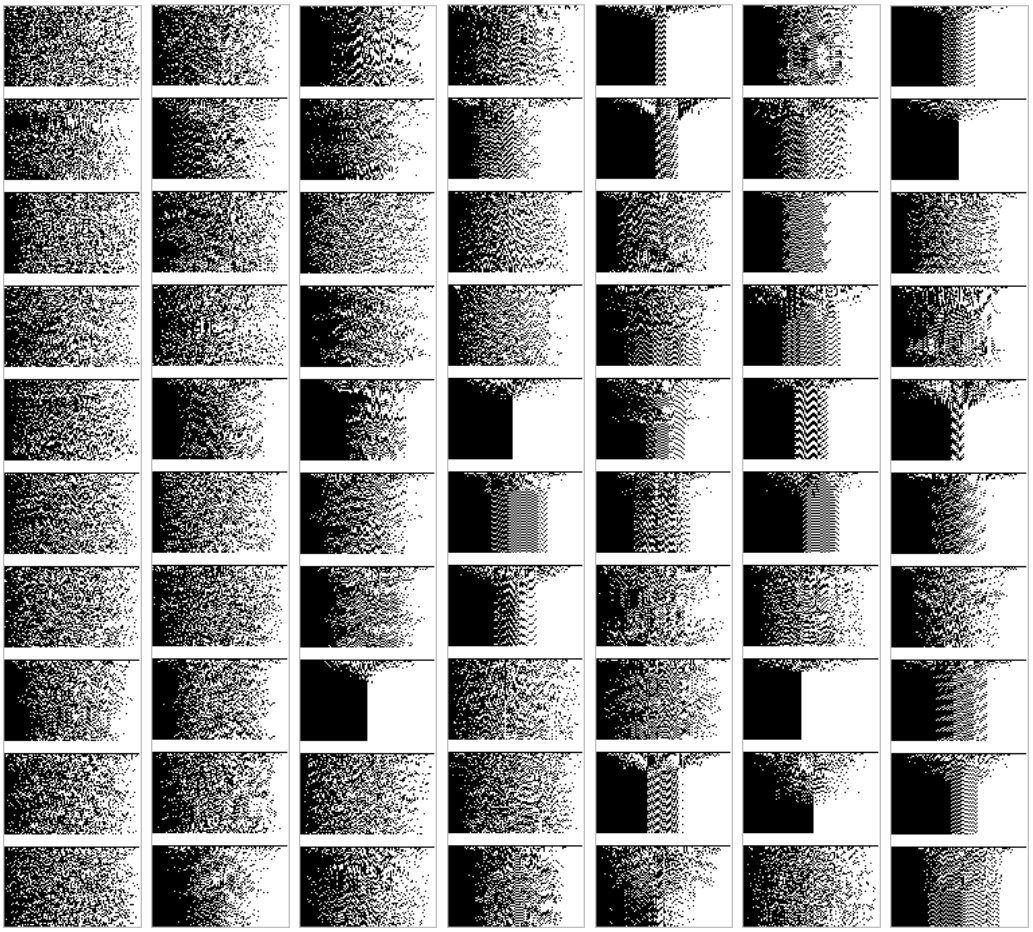

Figure 7.4 Visualisation of the time evolution of typical $K = 3$ RBNs, with $N = 100$, $t = 80$, initial condition all nodes "on"; columns have an increasing amount of canalisation, with the following number of canalised nodes: (a) 47 (b) 64 (c) 84 (d) 90 (e) 92 (f) 95 (f) 100

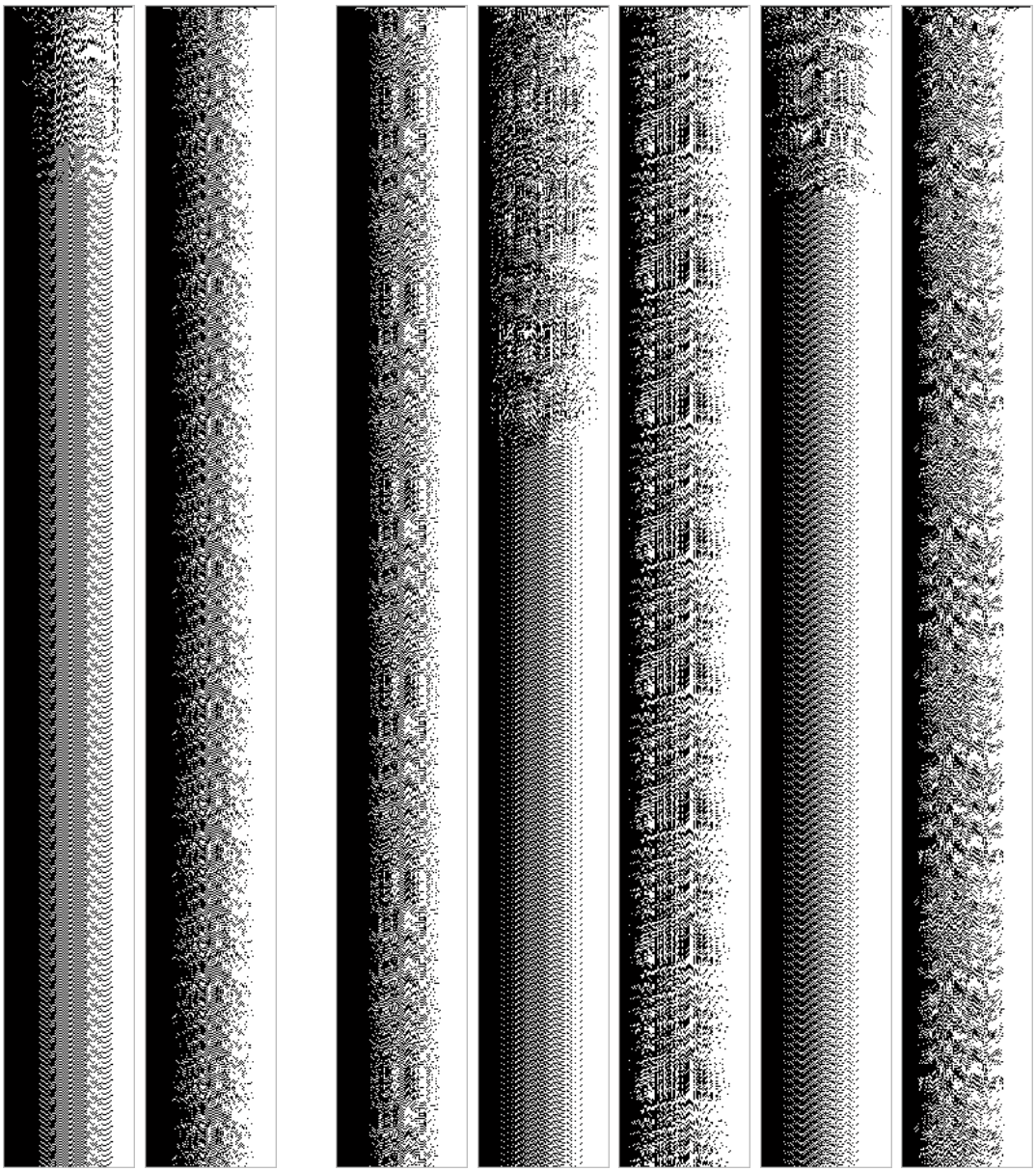

Figure 7.5 Visualisation of the time evolution of typical long cycle $K = 3$ RBNs, with $N = 100$, $t = 900$, initial condition all nodes "on", all nodes canalising. Left block of two runs, 84 nodes canalised; right block of five runs, all 100 nodes canalised.

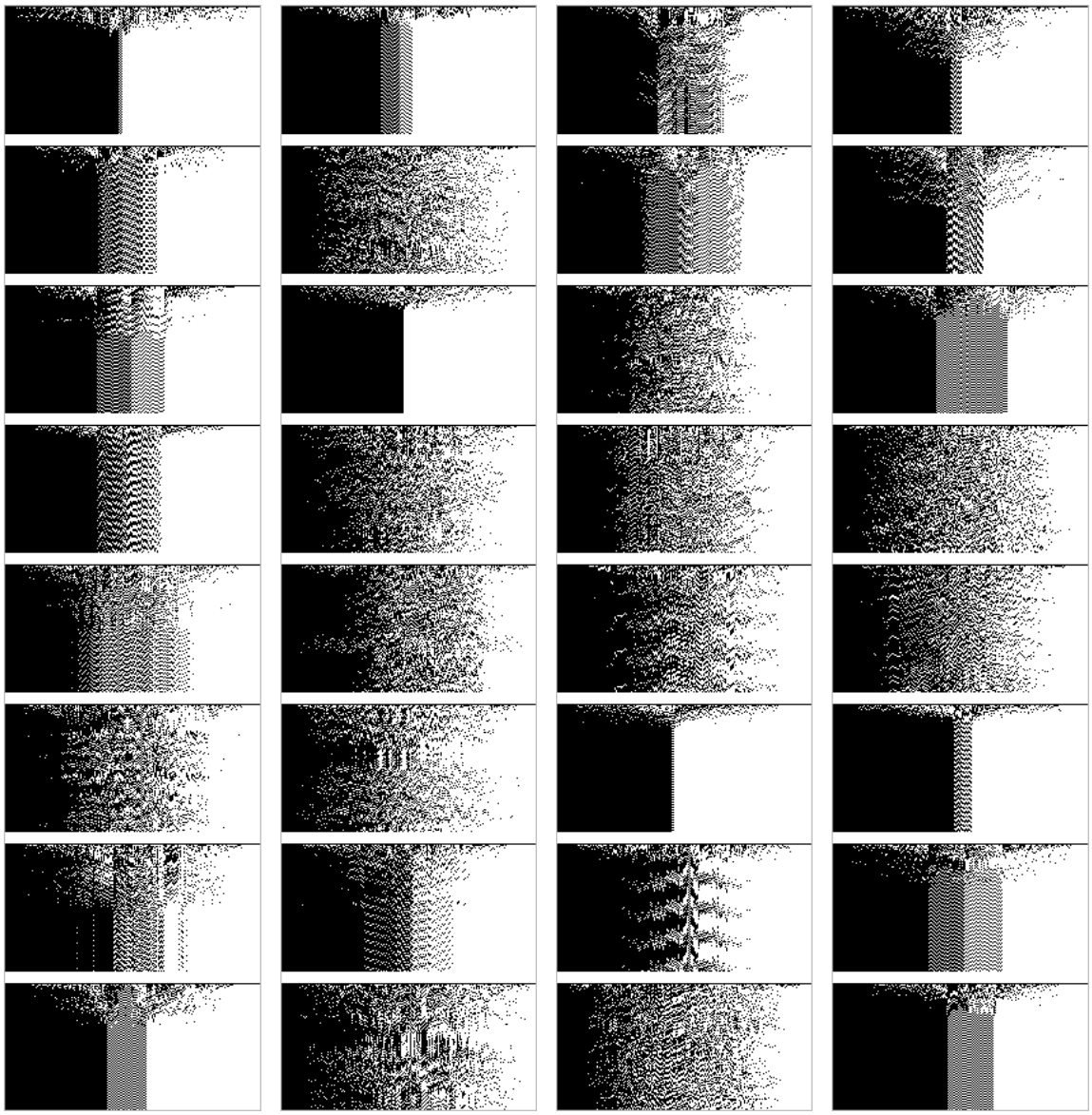

Figure 7.6 Visualisation of the time evolution of 32 typical $K = 3$ RBNs, with $N = 200$, $t = 80$, initial condition all nodes "on", all nodes canalising.

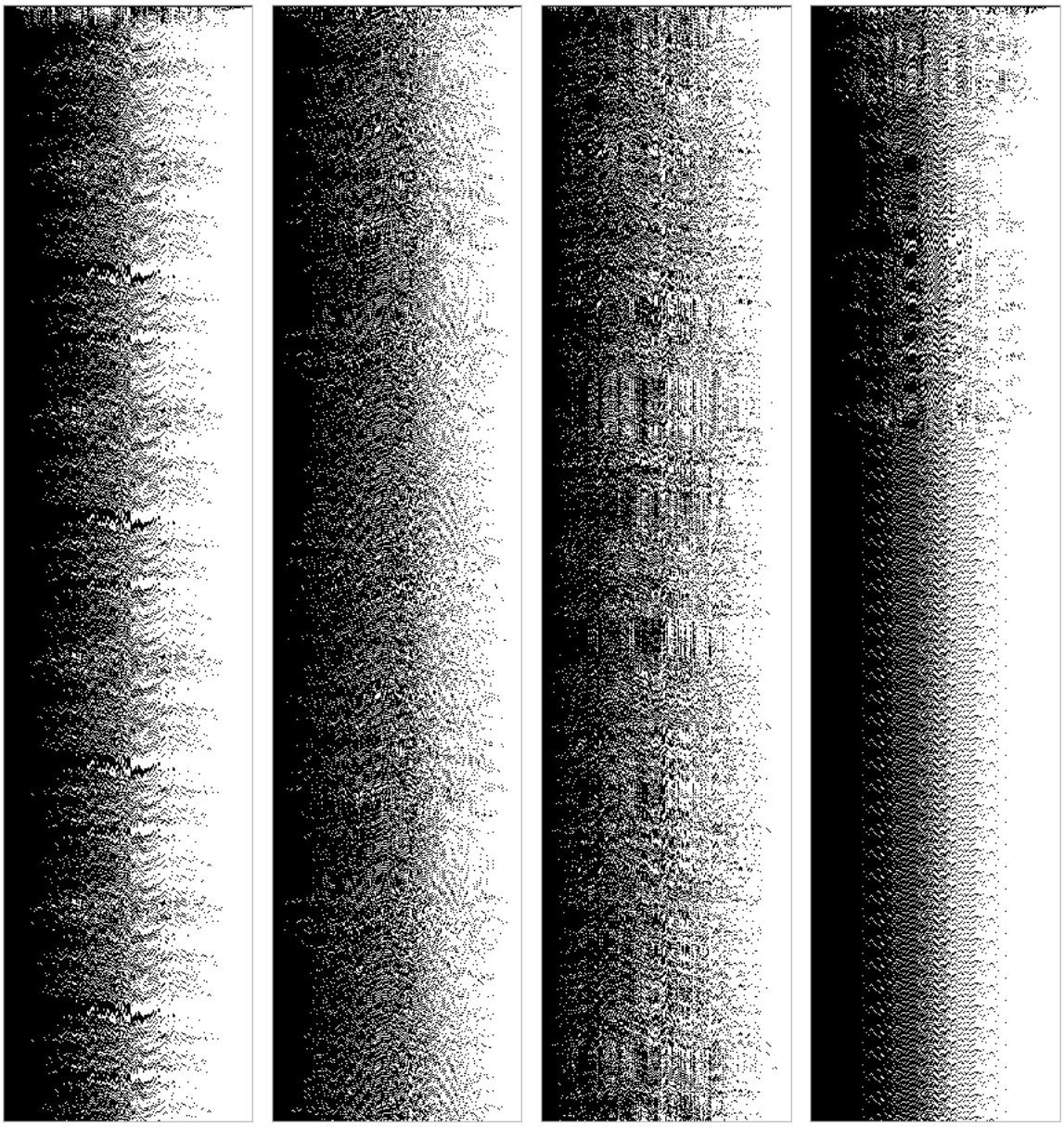

Figure 7.7 Visualisation of the time evolution of typical long cycle $K = 3$ RBNs, with $N = 200$, $t = 900$, initial condition all nodes "on", all nodes canalising.

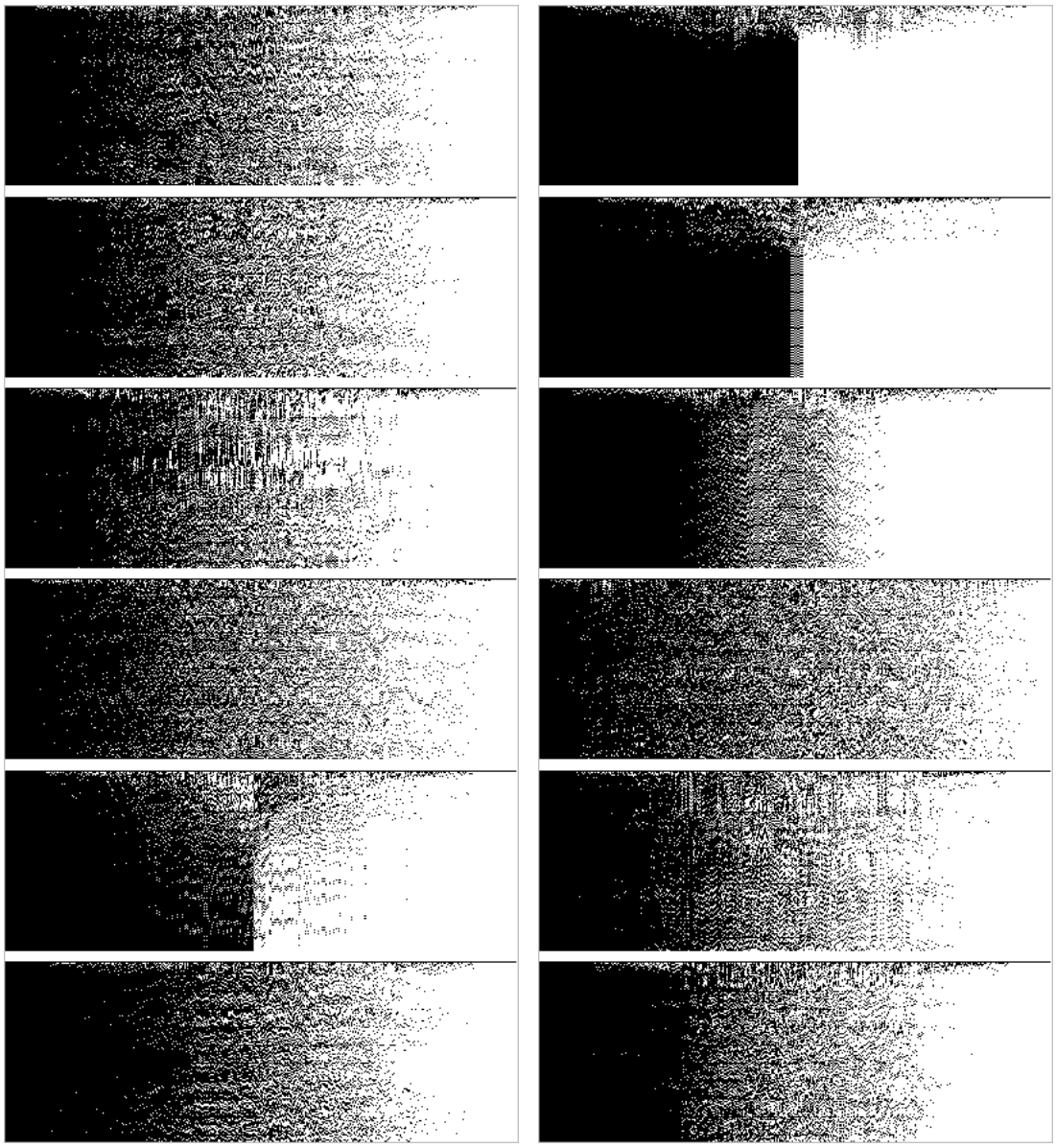

Figure 7.8 Visualisation of the time evolution of 12 typical $K = 3$ RBNs, with $N = 400$, $t = 150$, initial condition all nodes "on", all nodes canalising.

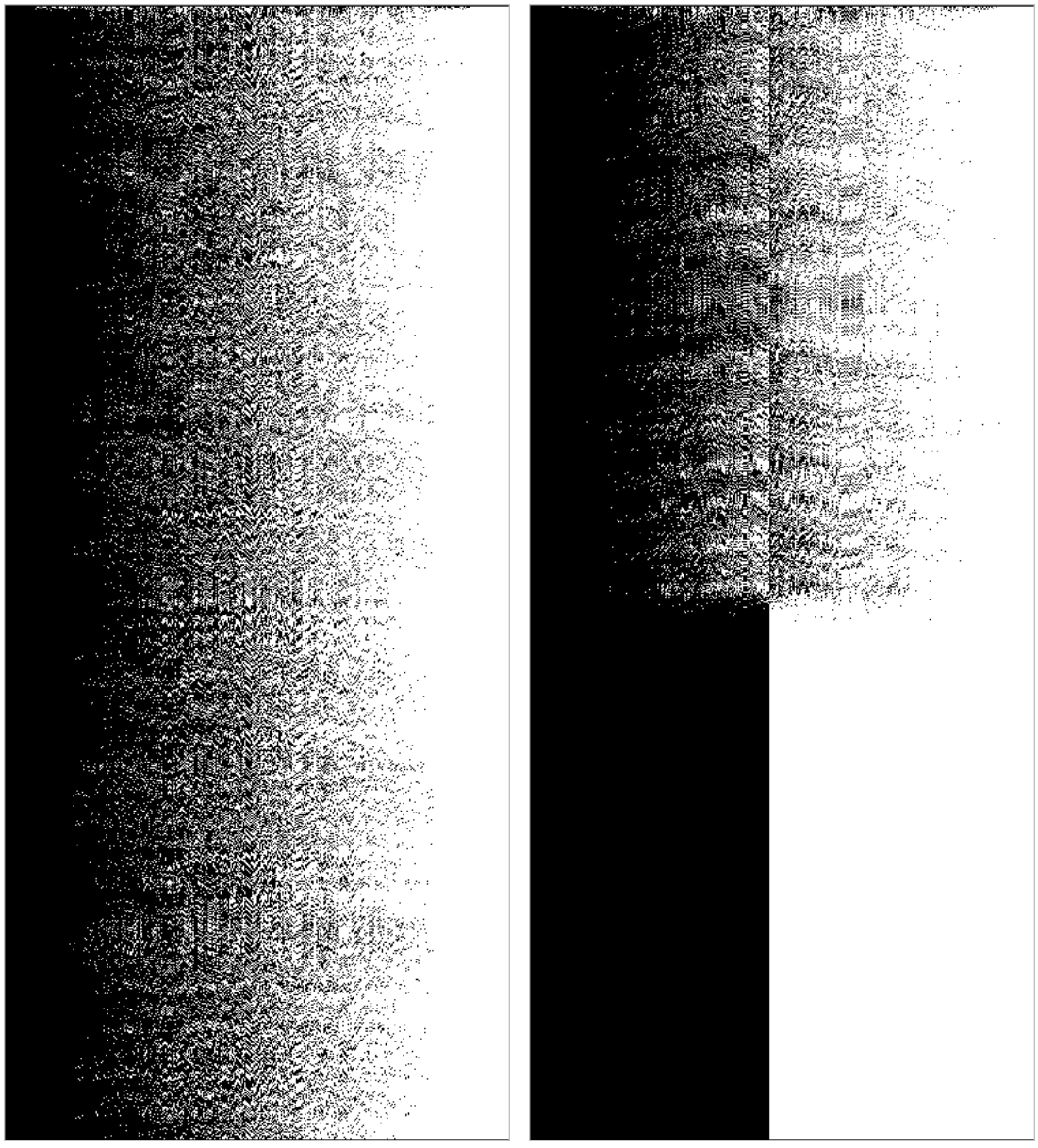

Figure 7.9 Visualisation of the time evolution of typical long cycle $K = 3$ RBNs, with $N = 400$, $t = 900$, initial condition all nodes "on", all nodes canalising.

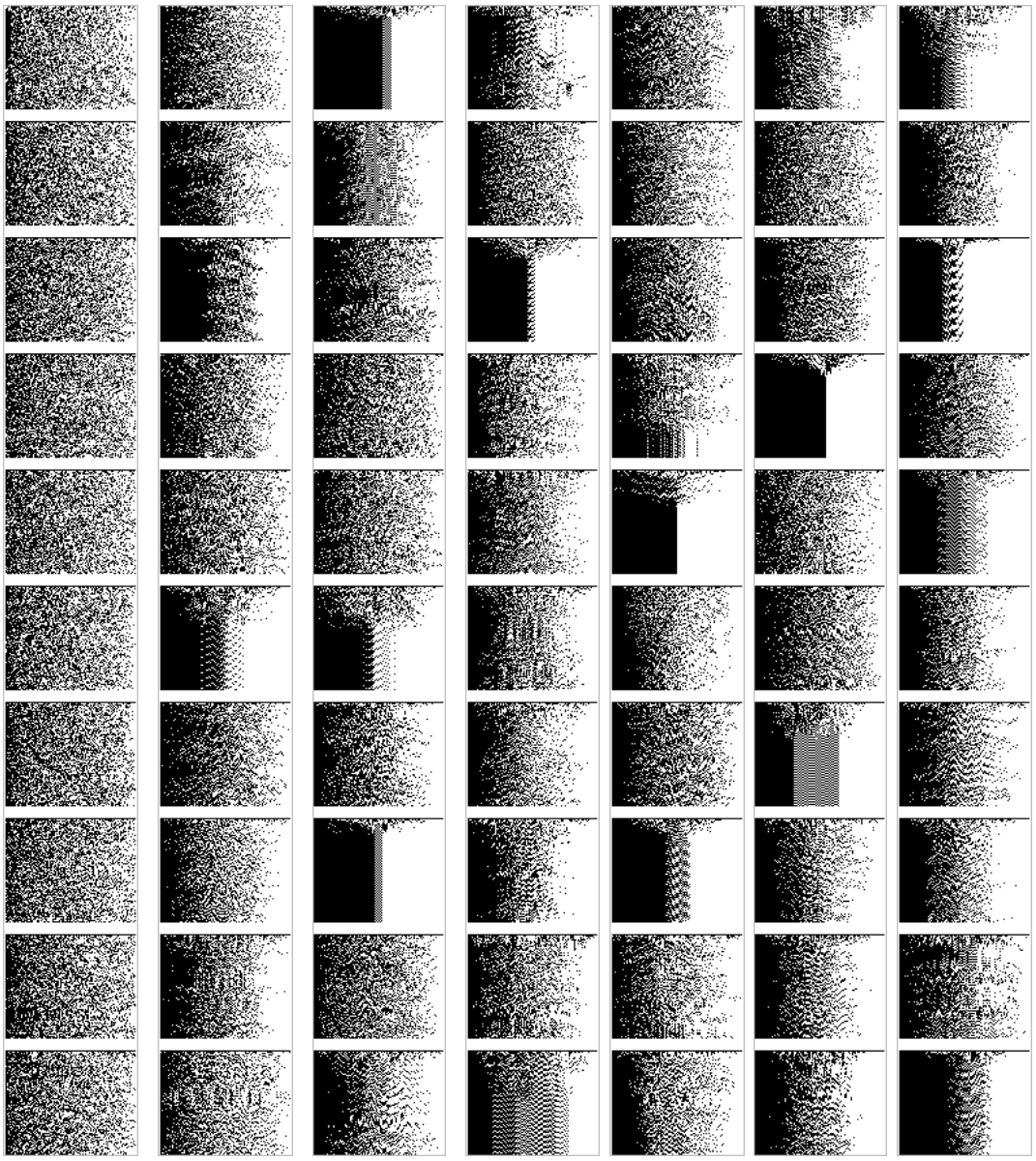

Figure 7.10 Visualisation of the time evolution of typical $K = 4$ RBNs, with $N = 100$, $t = 80$, initial condition all nodes "on"; columns have an increasing amount of canalisation, with the following number of canalised nodes: (a) 5 (b) 74 (c) 85 (d) 90 (e) 92 (f) 95 (f) 100

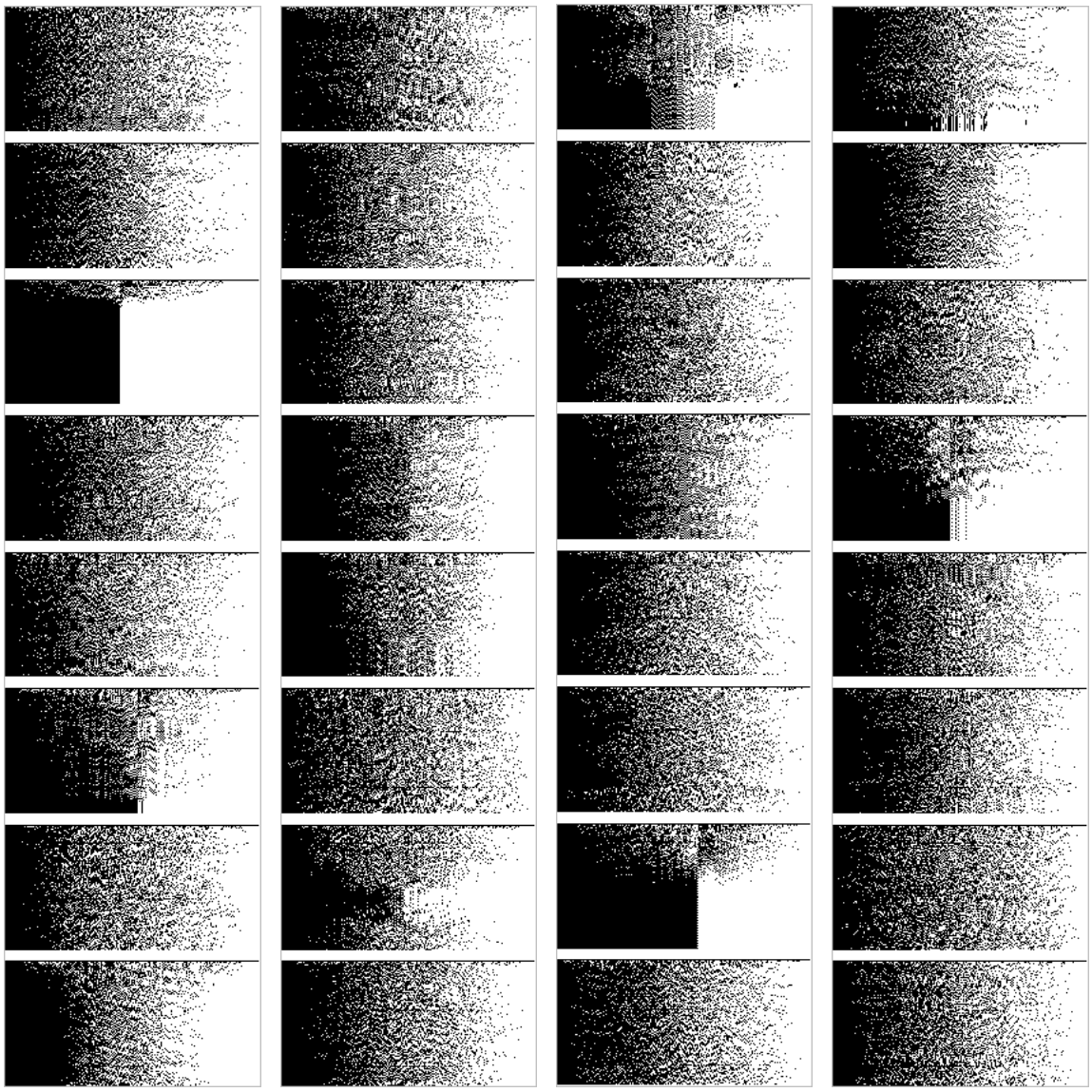

Figure 7.11 Visualisation of the time evolution of 32 typical $K = 4$ RBNs, with $N = 200$, $t = 100$, initial condition all nodes "on", all nodes canalising.

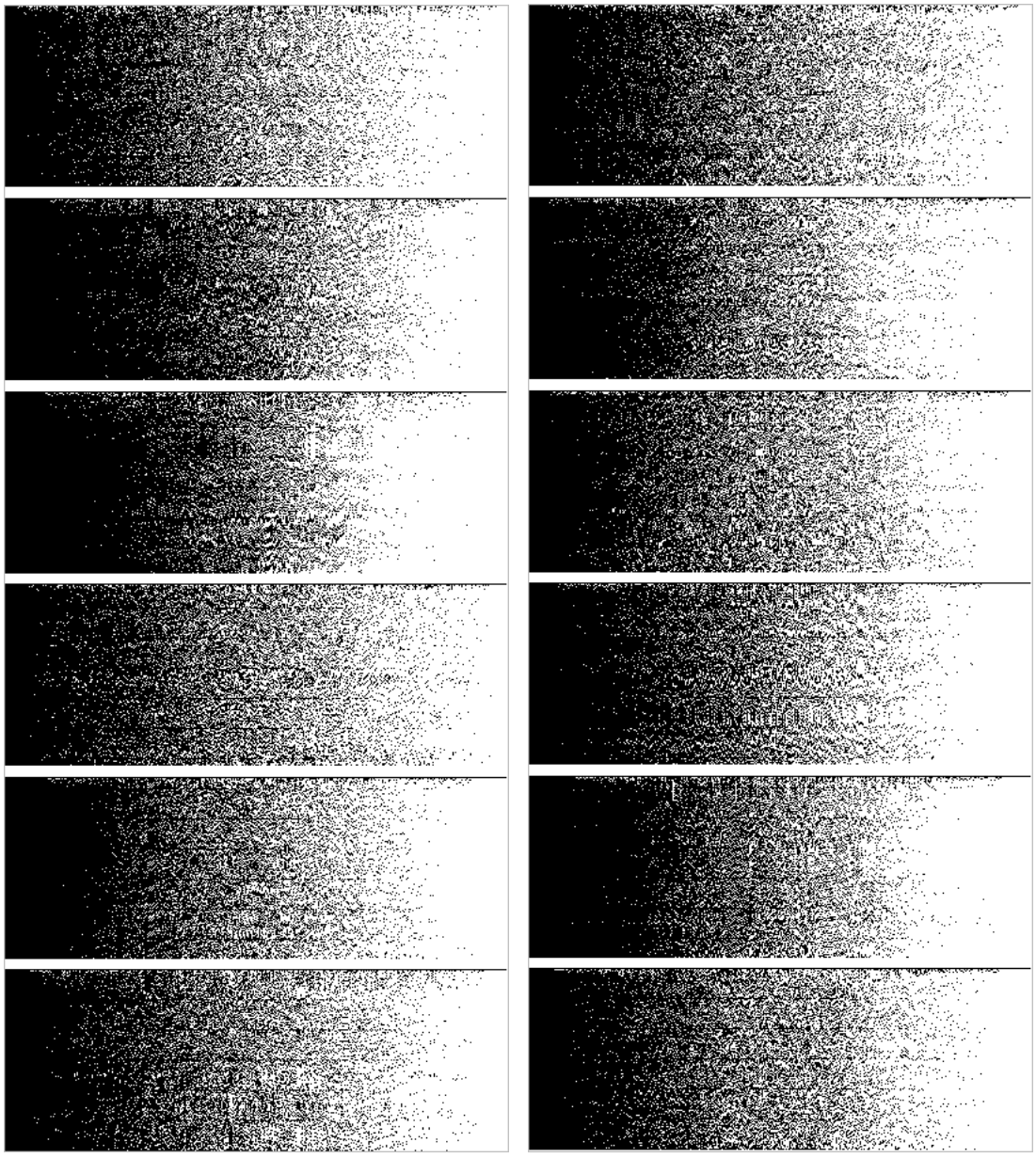

Figure 7.12 Visualisation of the time evolution of 12 typical $K = 4$ RBNs, with $N = 400$, $t = 150$, initial condition all nodes "on", all nodes canalising.

Chapter 8

Discussion and conclusions

We have introduced a very simple algorithm to allow the time behaviour of RBNs to be visualised in a manner that exposes the transient behaviour, and the structure of the frozen core and cycling nodes. The value of this approach has been demonstrated by showing various examples of the behaviour of RBNs as certain parameters are varied. The multiplicity of examples demonstrates the range of typical behaviours.

Visualisation of the dynamics helps to prime intuition, and to suggest hypotheses to explore. Some conjectures have been posed; more such conjectures could be generated from larger numbers of examples; some of these may be worthy of further investigation.

Acknowledgements

Thanks to Leo Caves, Adam Faulconbridge, Julian Miller, and Jon Timmis for comments on an earlier draft, and to Simon Poulding for detailed comments on the final draft.

Appendix A

Bibliography

[1] Barbara Drossel. Random boolean networks. In H. G. Schuster, editor, *Reviews of Nonlinear Dynamics and Complexity*, volume 1. Wiley, 2008. arXiv:0706.3351v2 [cond-mat.stat-mech].

[2] Carlos Gershenson. Introduction to random boolean networks. In *Workshop and Tutorial Proceedings, ALife IX*, pages 160–173, 2004.

[3] S. A. Kauffman. Requirements for evolvability in complex systems. *Physica D*, 42:135–152, 1990.

[4] S. A. Kauffman. *The Origins of Order*. Oxford University Press, 1993.

[5] Susan Stepney. Visualising random boolean network dynamics. In *GECCO 2009, Montreal, Canada, July 2009*, pages 1781–82. ACM Press, 2009.

[6] Susan Stepney. Visualising random boolean network dynamics: effects of perturbations and canalisation. In *ECAL 2009, Budapest, Hungary, September 2009*, LNCS. Springer, 2010.

[7] Edward R. Tufte. *The Visual Display of Quantitative Information*. Graphics Press, 1983.

[8] Stephen Wolfram. Random sequence generation by cellular automata. *Adv. in Appl. Math.*, 7:123–169, 1986.

[9] Andrew Wuensche. Genomic regulation modeled as a network with basins of attraction. *Pacific Symposium on Biocomput.*, pages 89–102, 1998.

[10] Andrew Wuensche and Mike Lesser. *The Global Dynamics of Cellular Automata*. Addison Wesley, 1992.

Appendix B

Matlab code

B.1 Sorting on the frozen core

```
s = 0;     % random seed, for reproducibility
rand('state', s);

k = 2;     % number of connections, K
n = 200;   % number of nodes, N
t = 100;   % number of timesteps
m = 8;     % number of runs

% the k random connections to each node n
%    generate n random permutations of length n;
%    select the first k items of each
[ignore,K] = sort(rand(n,n));
K = K(1:k,:);

% the random boolean function at each node n
B = dec2bin(floor(rand(1,n)*2^(2^k)))'-48;

Pow = 2.^sum(triu(ones(k),1));  % conversion function
%================================================
% initial conditions: 50% random initialisation
for i = 1:m
    X0(i,:) = rand(1,n); X0(i,:) = X0(i,:) < 0.5;
end

% alternative initial conditions: all on
X0(1,:) = ones(1,n);
```

```
%================================================
% sort the nodes

gr = 4;  % number of runs to sort on, <= m
Totals = zeros(n,1);
for j = 1:gr
    X = X0(j,:);
    for i = 1:t
        V = (Pow * X(K))+1;     % input value (k bits as number)
        X = diag(B(V,:));       % lookup boolean function
        Totals = Totals + X;
    end
end

[Totals,IX] = sort(Totals);     % sort the Totals, get the indices
B = B(:,IX);                    % permute the boolean function

K = K(:,IX);                    % permute the inputs, as sorted order
IXS = repmat(IX,[1 n k]);       % relabel the permuted indices
KS = repmat(reshape(K',[1 n k]),[n 1 1]);
[I,Dummy] = find(IXS==KS);
K = reshape(I,[n,k])';

X0 = X0(:,IX);      % permute initial conditions,
                    % so later using the same ones as for sorting
%================================================
% plot m runs of same RBN

Img = zeros(n,m*t);

for j = 1:m
    X = X0(j,:);

    Img(:,((j-1)*t)+1) = X;
    for i = 1:t
        V = (Pow * X(K))+1;     % input value (k bits as number)
        X = diag(B(V,:));       % lookup boolean function

% <==== perturbation code (see later) goes here ====>
```

```
        Img(:,((j-1)*t+i)+1) = X;
    end
end
imshow(im2bw(mat2gray(-Img)),'Border','tight');
```

B.2 Plotting the boolean function order

The scatter plots in figure 3.5 are plotted before (open circles) and after (filled circles) permuting the boolean functions:

```
hold on;
scatter([1:n], sum(B), 3, 'k');
B = B(:,IX);                    % permute the boolean function
scatter([1:n], sum(B)+0.1,  3, 'k', 'filled');
```

B.3 Perturbing the state

B.3.1 Near the centre

Introduce a state perturbation somewhere in the middle 10% of the nodes, every 20 iterations.

```
if (i > 99 && mod(i,20) == 0 )
    nn = n/2 + ceil(rand*n/10 - n/20);
    X(nn) = ~X(nn);
end
```

B.3.2 In the frozen core

Introduce a state perturbation somewhere in the outer thirds of the frozen core, every 20 iterations.

```
if (i > 99 && mod(i,20) == 0 )
    nn = mod(ceil(rand*n*2/3 - n/3) , n) + 1;
    X(nn) = ~X(nn);
end
```

B.4 Mutating the wiring

Introduce a wiring mutation somewhere in the middle 10% of the nodes, every 20 iterations. The mutation is made to input wire 1. The resulting mutation is required to result in a legal RBN: the new input node $k1$ must still be distinct from wire 2's input node $k2$ (this assumes a $K = 2$ network). However, the mutation is allowed to be "neutral", that is, to leave $k1$ unchanged.

```
if (i > 99 && mod(i,20) == 0 )
    nn = n/2 + ceil(rand*n/10 - n/20);
    k2 = K(2,nn);
    k1 = ceil(rand*n);
    while (k1 == k2)
        k1 = ceil(rand*n);
    end
    K(1,nn) = k1;
end
```

B.5 Mutating the boolean function

Introduce a boolean function mutation somewhere in the middle 10% of the nodes, every 20 iterations. The mutation is allowed to be "neutral", that is, to leave the function unchanged.

```
if (i > 99 && mod(i,20) == 0 )
    nn = n/2 + ceil(rand*n/10 - n/20);
    B(:,nn) = dec2bin(floor(rand*2^(2^k)),2^k)'-48;
end
```

B.6 Canalisation

B.6.1 Find the canalised functions, $K = 3$ example

```
function ans = is_canalised3

% k = connectivity = 3 = number of inputs to bool fn
% return boolean array of size 2^2^3 = 256
%   entries: 1 = canalised; 0 = not canalised
```

```
B = dec2bin([0:2^(2^3)-1])'-48;     % the 2^2^3 boolean functions

B10 = B([1:4],:); % 1st input 0
B11 = B([5:8],:); % 1st input 1
B20 = B([1,2,5,6],:); % 2nd input
B21 = B([3,4,7,8],:);
B30 = B([1,3,5,7],:); % 3rd input
B31 = B([2,4,6,8],:);

not(any(B10,1)) | all(B10,1) | not(any(B11,1)) | all(B11,1) ...
| not(any(B20,1)) | all(B20,1) | not(any(B21,1)) | all(B21,1) ...
| not(any(B30,1)) | all(B30,1) | not(any(B31,1)) | all(B31,1) ;
```

B.6.2 Initialise boolean function array, $K = 3$

```
% the random boolean function at each node n
% replace the line ...
% B = dec2bin(floor(rand(1,n)*2^(2^k)))'-48;
% by ...

% numbers of canalised functions for k=3
Canalised = find(is_canalised3)-1;
b_can = size(Canalised,2);
% numbers of non-canalised functions for k=3
NonCanalised = setdiff([0:255],Canalised);
b_non = 256 - b_can;

% of the n nodes, proportion p are canalised
p = 0.92;
n_can = ceil(p * n);
n_non = n - n_can;

% the random boolean function at each point n
Bcan = dec2bin(Canalised(ceil(rand(1,n_can)*b_can)),8)'-48;
Bnon = dec2bin(NonCanalised(ceil(rand(1,n_non)*b_non)),8)'-48;
B = [Bcan, Bnon];
```