The University of Manchester

The Effects of State Dependent and State Independent Probabilistic Updating on Boolean Network Dynamics

Natasha Saint Savage

2005

MIMS EPrint: **2006.32**

Manchester Institute for Mathematical Sciences
School of Mathematics

The University of Manchester

Reports available from: http://www.manchester.ac.uk/mims/eprints

And by contacting: The MIMS Secretary
School of Mathematics
The University of Manchester
Manchester, M13 9PL, UK

ISSN 1749-9097

The Effects of State Dependent and State Independent Probabilistic Updating on Boolean Network Dynamics

A thesis submitted to The University of Manchester for the degree of PhD in the Faculty of Engineering and Physical Sciences

2005

Natasha Saint Savage
School of Mathematics

Contents

Word Count: 22201

List of Figures

List of Tables

THE UNIVERSITY OF MANCHESTER
ABSTRACT OF THESIS
submitted by Natasha Saint Savage
for the degree of PhD and entitled

The Effects of State Dependent and State Independent Probabilistic Updating on Boolean Network Dynamics.

September 2005

We study semi-synchronous Boolean networks with probabilistic updating schemes and various topologies (tree, loop, and random). As well as state independent probabilistic updating we investigate a state dependent scheme which allows us to control the 'accuracy' of nodes. A node is accurate at n if it has been updated at n, or if its state is as it would be if it had updated.

The state dependent re-evaluation probabilities are determined by the 'accuracy heuristic': a stochastic equation which depends on the estimation of a distribution; we look at ways of estimating this distribution and derive variance expressions for the estimators.

Through our work on random Boolean trees we observe that (in general) the output of a Boolean function with correlated inputs, becomes less correlated as the number of inputs is increased. We also discover that the correlation of a Boolean function's output directly affects the ability of the heuristic to achieve the node's target accuracy.

Deterministic random Boolean network dynamics are viewed in a new way, via the distribution of node output distributions (the probability a node's state is 1 or 0). This view shows the 'activity' of nodes across the network. We find that as in-degree is increased the topology has less effect on the activity and the distribution of the Boolean functions dominates. We present a theoretical result to support this theory.

To understand the dynamics of probabilistically updating Boolean networks we use a numerical approximation to Flyvbjerg's frozen component.

The concept of stability in probabilistically updating Boolean networks is addressed and investigated. For the loop topology the dynamics of active loops fall into two categories: those with an odd number of inversion nodes and those with an even number. We discuss the stability of a fixed point in both cases. For the random topology we derive an annealed approximation which indicates a phase transition similar to that previously found in the deterministic networks.

Declaration

I declare that no portion of the work referred to in the thesis has been submitted in support of an application for another degree or qualification of this or any other university or any other institute of learning.

Copyright Statement

Dedicated to Anna Savage

Who gave me a beautiful world to live in.

Acknowledgements

Dr Jerry Huke

Gratitude and admiration goes to Dr Jerry Huke. I wish to thank him for engaging in a project quite different from his own research interests, introducing and pursuing avenues of exploration that have molded this thesis. For taking the time to listen to my ideas, consider them and respond with insightful comment. My thanks to Jerry for an enjoyable, stimulating and productive experience.

Dr Keith Briggs

Warm thanks to Dr Keith Briggs for his friendship and continuing support of my scientific career. For his assistance in helping me design a PhD project, commenting on my reports and having a mind to send me relevant papers. For his invaluable introduction to the scientific community in general; inviting me to accompany him and attend meetings, introducing me to fellow scientists and giving me the opportunity to work along side him at British Telecom.

Chapter 1

Introduction: Random Boolean Networks

In 1969 Stuart Kauffman wrote a compelling paper that showed random Boolean networks (RBNs) to be a useful tool in modelling genetic nets [11]. The most surprising result at the time was what he went on to call 'order for free' [14]: Kauffman found that a dynamical system consisting of N coupled randomly chosen Boolean functions, each with two inputs, split the state space into a small number of short cycles (around $\sqrt{N}$ cycles, approximately $\sqrt{N}$ in length). RBNs with in-degree greater than 2 do not share these properties. Kauffman's work headed a surge of research activity and a great number of papers have been, and are still being, written as a result. In the following sections we will introduce the RBN and describe a few results which have been chosen for their relevance to this thesis.

1.1 Deterministic RBN's

1.1.1 Mathematical setup for deterministic RBNs

RBN's are networks consisting of N 'nodes' (typically represented by small circles or dots) where each node can take on the values 0 or 1. (These values collectively form the state of the network; we also refer to the value adopted by a particular node as the state of the node.) Each node is connected to other nodes; in particular, each node has k 'predecessors', nodes which provide inputs to it, in the sense that its state at time $n+1$ is determined by the states of the predecessors at time n. (k is the same for all nodes but the k predecessors are not necessarily all distinct, and a node can be among its own predecessors.) Pictorially, the fact that node A is a predecessor to node B is represented by a directed curve from A to B. Thus each node has k incoming curves and a variable number of

outgoing curves, depending on how many other nodes have it as a predecessor, no particular restrictions are placed on how many nodes a given node can output to. A graph representing the input/output relationships of the nodes is called a 'connection graph' of the network.

Each node has a Boolean function $f : \{0,1\}^k \to \{0,1\}$ which computes the state of a node at time $n+1$ from the states (at time n) of the k predecessors; but these functions are not usually represented pictorially.

A complete specification of the network is made by specifying, for each node, which k nodes form its predecessors, and what Boolean function links its state to its predecessors states. The connections and Boolean functions constitute the 'architecture' of the network. Kauffman suggested that the architecture be chosen at random (hence the name 'random Boolean network'), and the idea was then to investigate the behaviour of a typical network. (Note that once the architecture is chosen the time evolution of the network is completely deterministic.)

There are 2^{2^k} possible Boolean functions of k variables; an example of such a set (for $k = 2$) is shown in table 1.1. Each node in the network is assigned one of these, and the predecessors can be assigned in $N!/(N-k)!$ different (ordered) ways. Thus the number of possible architectures is [9],

$$\left(\frac{2^{2^k} N!}{(N-k)!} \right)^N$$

Many of these networks will be equivalent in the sense that they will show identical behaviour given the same initial conditions; for example, if a node is assigned one of the two constant Boolean functions its predecessors can be changed without effecting the dynamical behaviour in any way. Nevertheless it is clear that the number of RBNs for given N, k can be extremely large.

The classic RBN (as introduced by Kauffman) is a dynamical system with states $X^n = x_0^n x_1^n x_2^n \ldots x_{N-1}^n \in \{0,1\}^N$, governed by a deterministic map, $F : \{0,1\}^N \to \{0,1\}^N$. F consists of a system of Boolean functions, $f : \{0,1\}^k \to \{0,1\}$,

$$\begin{aligned}
x_0^{n+1} &= f_0(x_{0,0}^n, x_{0,1}^n, \ldots, x_{0,k-1}^n) \\
x_1^{n+1} &= f_1(x_{1,0}^n, x_{1,1}^n, \ldots, x_{1,k-1}^n) \\
\vdots \quad &\quad \vdots \\
x_{N-1}^{n+1} &= f_{N-1}(x_{N-1,0}^n, x_{N-1,1}^n, \ldots, x_{N-1,k-1}^n)
\end{aligned}$$

$x_{i,0}$	$x_{i,1}$	f_i^0	f_i^1	f_i^2	f_i^3	f_i^4	f_i^5	f_i^6	f_i^7
0	0	0	1	0	1	0	1	0	1
0	1	0	0	1	1	0	0	1	1
1	0	0	0	0	0	1	1	1	1
1	1	0	0	0	0	0	0	0	0

$x_{i,0}$	$x_{i,1}$	f_i^8	f_i^9	f_i^{10}	f_i^{11}	f_i^{12}	f_i^{13}	f_i^{14}	f_i^{15}
0	0	0	1	0	1	0	1	0	1
0	1	0	0	1	1	0	0	1	1
1	0	0	0	0	0	1	1	1	1
1	1	1	1	1	1	1	1	1	1

Table 1.1: The truth table showing the finite set of all Boolean functions for $k = 2$. f_i^j is the Boolean function that gives a truth table according to the 2^k binary expansions of j, (top to bottom).

$x_{i,j}$ is the j'th variable of f_i, $\quad x_{i,j}^n \in \{x_0^n, x_1^n, x_2^n, \ldots, x_{N-1}^n\}$. An example of F for $N = 4$ is given by map (1.1).

$$\begin{aligned} x_0^{n+1} &= f_0(x_1^n, x_0^n) \\ x_1^{n+1} &= f_1(x_2^n, x_0^n) \\ x_2^{n+1} &= f_2(x_1^n, x_3^n) \\ x_3^{n+1} &= f_3(x_1^n, x_2^n) \end{aligned} \tag{1.1}$$

where $f_0, \ldots, f_3$ are Boolean functions of two arguments.

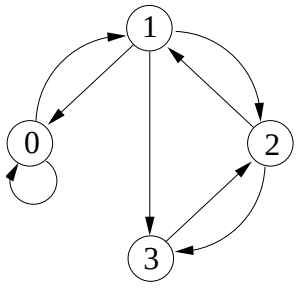

Figure 1.1: Connection graph of the classic RBN described by (1.1)

As there are finitely many states (2^N) in Boolean state space of a given network, we can easily show the dynamical behaviour by a 'phase portrait'; this consists of a graph whose vertices represent the 2^N states, and whose (directed) edges show how each state evolves in one time step (thus there is an edge from X to Y if and only if $F(X) = (Y)$). (This phase portrait is sometimes called the 'iteration graph' of the network.) As an example let $f_0 = f_0^2$, $f_1 = f_1^5$, $f_2 = f_2^7$

and $f_3 = f_3^9$, in map (1.1). Perform F on each of the 16 states ($= 2^4$) (as shown in table 1.2) giving the phase portrait shown in figure 1.2.

X^n	0000	0001	0010	0011	0100	0101	0110	0111
X^{n+1}	0111	0111	0110	0110	0110	0100	0111	0101
X^n	1000	1001	1010	1011	1100	1101	1110	1111
X^{n+1}	1011	1011	1010	1010	0010	0000	0011	0001

Table 1.2: One iterate of F on each state $X^n \in \{0,1\}^4$.

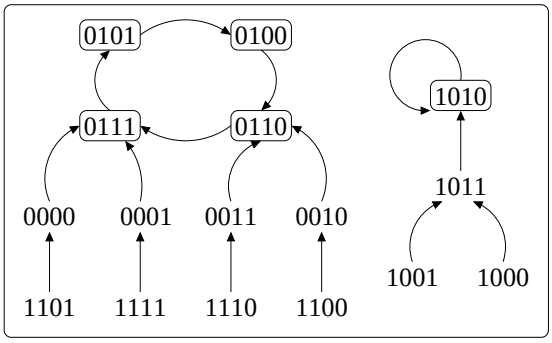

Figure 1.2: Phase portrait of F (given in table 1.2).

The phase portrait shows that F splits the state space into two periodic orbits, a period 4 and a period 1 (a period 1 orbit is also called a fixed point). We will term the states leading to a periodic orbit and the orbit itself as a basin of attraction. The classic RBN splits the state space into disjoint basins of attraction, thus long term behaviour of a deterministic RBN depends on the initial state (once F is fixed).

Most physical systems are subject to noise so it is important to analyze the stability of periodic orbits. In general, periodic behaviour is considered unstable if small perturbations tend to grow (and stable if they do not). This requires some notion of distance between states; the most commonly used distance for Boolean state space is the normalized Hamming distance,

$$d(X,Y) = \frac{1}{N}\sum_{i=0}^{N-1} |x_i - y_i|$$

Kauffman investigated the stability of periodic orbits by considering noise of the order $1/N$, that is he considered the trajectories of states a normalized Hamming distance $1/N$ apart. He recorded his observations in the form of transition matrices. To illustrate, label the fixed point in map F as cycle 1 and the period 4

cycle, cycle 2, the cycle transition matrix for map F is,

$$A = \begin{pmatrix} 0.5 & 0 \\ 0.5 & 1 \end{pmatrix}$$

The element $a_{i,j}$ of A is the probability of going from cycle j to i when a state in cycle j is subjected to random $1/N$ noise. As the network has only 4 nodes we can draw its state space as a 'torus' (figure 1.3, opposite edges (top and bottom, left and right) should be considered joined together). States with touching edges have $d = 1/4$. The heavy rectangle on the upper left indicates the basin of attraction of the fixed point; all other states are in the basin of the periodic cycle. This representation of state space makes calculating A straightforward and clearly illustrates the stability of cycle 2.

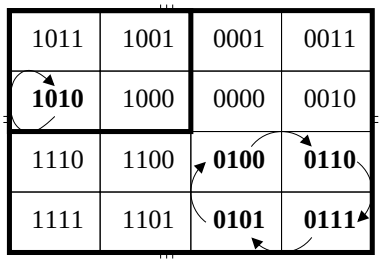

Figure 1.3: State space $\{0,1\}^4$ drawn as a torus, giving a geometric view of the Hamming distance.

Kauffman generated transition matrices for maps with N up to 2000 and different values of k. He noted that for maps whose Boolean functions depended on two variables, 95% of perturbations returned to their original orbit. Kauffman termed this 'homeostatic stability' [12]. The $k = 2$ orbits also had restricted local reachability, meaning that any one perturbed cycle could reach 6 others at most. To put this into perspective, a RBN with $N = 2000$ would be expected to have 45 periodic orbits [14]. Kauffman found for $k > 2$ the homeostatic stability of periodic orbits becomes less frequent and reachability increases. Table 1.3 summarizes the classic RBN properties observed by Kauffman [14].

1.1.2 Dynamics of deterministic RBNs

The state space of $k = 2$ classic RBNs consists of few short cycles embedded in large basins of attraction (giving them their (homeostatic) stability), and as k is increased the cycles become longer and the basins smaller (see table 1.3). Kauffman and others observe a further difference between networks with high and low k values. For $k = 2$ networks, once the long term (periodic) behaviour has been

	Median state cycle length	Number of state cycle attractors	Homeostatic stability	Reachability among cycles after perturbation
$k = N$	$0.5 \times 2^{N/2}$	N/e	Low	High
$k > 5$	0.5×2^{BN} $B > 1$	$\sim N^{\left[\frac{log\left(\frac{1}{1/2 \pm \alpha}\right)}{2}\right]}$ $\alpha = p_k - 1/2$	Low	High
$k = 2$	$\sqrt{N}$	$\sqrt{N}$	High	Low
$k = 1$	$(\pi/2)^{1/2}(N^{1/2})$	Exponential in N	Low	High

Table 1.3: Summary of the properties of classic RBNs observed by Kauffman, $p_k = \Pi_{i=1}^{k}(1 - i/(2^N))$ [14]

established, most nodes are fixed in states which do not change with time (we say such nodes are 'frozen'); only a relatively small proportion show non-constant behaviour (are 'active') [11] [7] [2]. For high values of k ($k \geq 5$) a high proportion of nodes are active even when the periodic orbit is reached. Intermediate values of k show more complex behaviour with more parts of the network highly active, other parts showing periods of low activity.

The first theoretical work to shed more light on this phenomenon was that of Derrida and Pomeau [6]. They compared the evolution of two states of the network, which they supposed to be a normalized Hamming distance d_n apart at time n and derived an approximate expression for d_{n+1} by averaging over all networks with fixed N, k. Effectively, they examined the behaviour of a network whose nodes do not change but whose architecture is chosen randomly on every iteration. Obviously this is different to the classic RBN, where architecture is fixed; Derrida and Pomeau call their version an 'annealed approximation'.

The derivation of Derrida and Pomeau's expression is clearer if we consider the normalized overlap of two states, $a_n = 1 - d_n$, rather than the distance [22]. Let the 'overlap region' be the set of nodes that have the same value in the two states; a_n is just the fraction of nodes in the overlap region at n. What is the value of a_{n+1}? Node i will certainly be in the overlap region at $n + 1$ if all its predecessors are in the overlap region at n. But because these predecessors are randomly assigned in the annealed approximation the probability of this is just a_n^k. If one

or more of the predecessors lies outside the overlap region (which happens with probability $1 - a_n^k$) some of the values $x_{i,j}^n$ $(j = 0, 1, \ldots, k-1)$ are different in the two states, and node i will then be in the overlap region at $n+1$ only if the values $f_i(x_{i,0}, \ldots, x_{i,k-1})$ are the same for the two states despite these differences. Since the function f_i is assigned randomly, the probability of this is $1/2$. So,

$$\begin{aligned} a_{n+1} &= a_n^k + \frac{1}{2}(1 - a_n^k) \\ &= \frac{1}{2}(1 + a_n^k) \end{aligned}$$

Substituting $a_n = 1 - d_n$ in the overlap equation we get,

$$d_{n+1} = \frac{1}{2}(1 - (1 - d_n)^k) \tag{1.2}$$

Figure 1.4 shows Derrida and Pomeau's equation (equation (1.2)) for $k = 1, 2, 3, 4$, and $d_n = d_{n+1}$. From graphical analysis the fixed point at the origin is stable for $k \leq 2$ and unstable otherwise. The transition point (from stable to unstable) is confirmed by the gradient of equation (1.2) at the origin.

$$\frac{d}{dd_n} d_{n+1} \bigg|_{d_n=0} = \frac{k}{2}$$

The gradient is 1 at the transition point so $k = 2$ is a critical value (for $k \leq 2$ close trajectories converge, $k > 2$ close trajectories are drawn to the other fixed point, $d_n = d_{n+1} = 1 - (2/k)^{(k-1)/2}$).

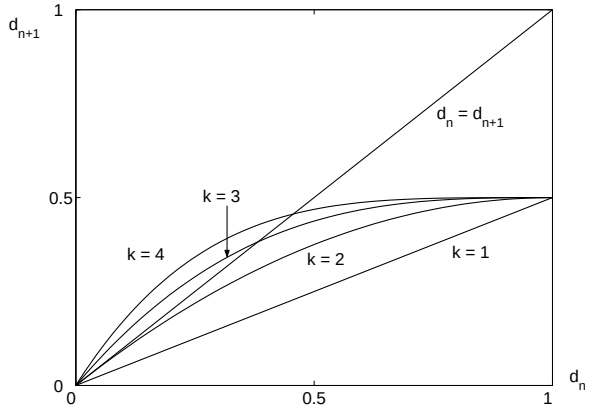

Figure 1.4: Graph of equation (1.2) for $k = 1, 2, 3, 4$, and $d_n = d_{n+1}$.

Derrida and Pomeau's argument does not provide a complete explanation for the transition observed in classic RBNs because it assumes that the behaviour of d_n as n increases can be found by iterating equation (1.2). But this equation can

only be iterated if the annealed approximation holds, that is if the architecture is randomly reassigned each time step. It is clear that the annealed scheme will not show the long term periodic behaviour of the classic RBNs. The precise relationship between the annealed scheme and the classic one has in fact not been made clear up to now. Another approach to describing the transition is considered later.

Luque and Solé (1995) extended the annealed approximation to encompass generalized RBNs [23]. For a general RBN the number of predecessors is not fixed: node i has k_i predecessors where k_i is chosen randomly from some distribution. This distribution is fixed, say P_{k_i} is the probability that the number of predecessors is k_i. When defining a generalized RBN, k_i is set and then the Boolean function is chosen at random from the $2^{2^{k_i}}$ possibilities.

In the annealed approximation the distance equation for the generalized RBN is very similar to that of the classic RBN (equation (1.2)) only now we must sum over all k_i,

$$d_{n+1} = \frac{1}{2}(1 - \sum_{k_i=1}^{k_{max}} P_{k_i}(1-d_n)^{k_i}) \tag{1.3}$$

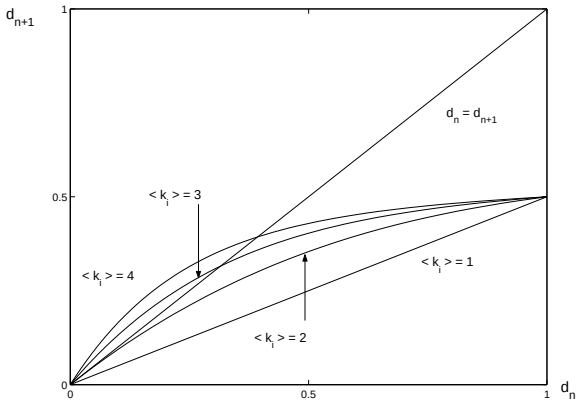

Figure 1.5: Graph of equation (1.3) for $\langle k_i \rangle = 1, 2, 3, 4$, and $d_n = d_{n+1}$.

In figure 1.5 we have drawn an example of equation (1.3) for k_i chosen randomly from the interval $[1, k_{max}]$, making $P_{k_i} = 1/k_{max}$ and the expectation of k_i (denoted $\langle k_i \rangle$) $(1+k_{max})/2$. As for the classic RBN, we have monotonically increasing curves, and the fixed point at the origin appears stable for $\langle k_i \rangle \leq 2$. To check the transition value of $\langle k_i \rangle$ differentiate equation (1.3) as before,

$$\frac{d}{dd_n} d_{n+1}\Big|_{d_n=0} = \frac{1}{2}\sum_{k_i=1}^{k_{max}} P_{k_i} k_i$$

for slope 1 we get,

$$\sum_{k_i=1}^{k_{max}} P_{k_i} k_i = \langle k_i \rangle = 2$$

So that if the distribution P_{k_i} has mean less than or equal to two we can expect stable dynamics.

In 2000 Luque and Solé used an annealed approximation (they called it the mean field approximation) to derive a discrete space-time Lyapunov exponent for classic RBN's, which also gave $k = 2$ as the transition point [15]. For a Boolean function $f : \{0,1\}^k \to \{0,1\}$ define the Boolean derivative at $X = x_0, x_1, \ldots x_{N-1} \in \{0,1\}^N$, with respect to x_l as,

$$\delta f_l(X) = f(x_0, \ldots, x_l, \ldots x_{N-1}) \oplus f(x_0, \ldots, \bar{x}_l, \ldots x_{N-1})$$

$\delta f_l(X)$ equals 1 if 'flipping' the value of x_l (denoted $\bar{x}_l$) from 0 to 1 or vice versa changes the value of $f(X)$; otherwise $\delta f_l(X) = 0$. As discussed earlier, the stability of a point (or orbit) can be investigated by changing one of the node values and seeing if this change grows. Luque and Solé proposed to do this by looking at the Boolean derivatives of the node functions f_i. Abbreviate $\delta f_{i,l}(X^n)$ by $\delta f_{i,l}(n)$, so that in the notation of subsection 1.1.1,

$$\delta f_{i,l}(n) = f_i(x^n_{i,0}, \ldots, x^n_{i,l}, \ldots x^n_{i,N-1}) \oplus f_i(x^n_{i,0}, \ldots, \bar{x}^n_{i,l}, \ldots x^n_{i,N-1})$$

Luque and Solé consider a perturbation to the state X^n, which they call D_n: D_n is a column vector of N binary elements, a one in position l signifying that the value of x^n_l is flipped (and 0 that it is not flipped). They then want to know about D_{n+1}, the resulting perturbation one time step later; in particular, is $|D_{n+1}|$ (where $|D_{n+1}|$ is the normalized Hamming size - or distance from $00\ldots0$) larger than $|D_n|$?

One can define a Jacobian, $\delta F(n)$, which records the effects on map F of all possible one variable flips.

$$\delta F(n) = \begin{pmatrix} \delta f^j_{0,0}(n) & \delta f^j_{0,1}(n) & \ldots & \delta f^j_{0,N-1}(n) \\ \delta f^j_{1,0}(n) & \delta f^j_{1,1}(n) & \ldots & \delta f^j_{1,N-1}(n) \\ \vdots & & & \\ \delta f^j_{N-1,0}(n) & \delta f^j_{N-1,1}(n) & \ldots & \delta f^j_{N-1,N-1}(n) \end{pmatrix}$$

Luque and Solé go on to define a Boolean matrix multiplication, $\odot$, which is basic matrix multiplication but every number greater than 1 is replaced with 1. This can be used to estimate D_{n+1} given $\delta F(n)$ and D_n,

$$D_{n+1} \approx \delta F(n) \odot D_n$$

To get a feel for this method generate $\delta F(n)$ for F given by equation (1.1) with $X^n = (1010)$ (the fixed point, figure 1.2). The first element of the Jacobian is, $\delta f^2_{0,0}(n) = f^2_0(\bar{x}^n_1, x^n_0) \oplus f^2_0(x^n_1, x^n_0) = 1$. Similar calculations for all elements of $\delta F(n)$ give,

$$\delta F(n) = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \end{pmatrix}$$

We can see from $\delta F(n)$ that the only way for $|D_{n+1}| > |D_n|$ is for $D_n = (1,0,0,0)^T$ or $(0,1,0,0)^T$. If we subject X^n to either $D_n = (1,0,0,0)^T$ or $(0,1,0,0)^T$ we get a state outside the fixed points basin of attraction (see figure 1.2).

In order for Luque and Solé to get a general result for the 'spread of damage' (i.e. growth of perturbations) with this method they estimated $\delta F(n)$ using the mean field approach. $\delta F(n)$ is a direct result of the Boolean functions that make up F, and X^n. If F is randomly redefined for every iteration (as it is in the mean field approach), a random matrix $\Omega(n)$ can replace $\delta F(n)$, the only constraints of the matrix are; it must have at most k ones per row, and for $\Omega(n)$ to represent choosing f^j_i and its variables randomly the mean number of ones per row is $k/2$. Using $\Omega(n)$ they defined the discrete Lyapunov exponent,

$$\lambda(T) \;=\; \frac{1}{T}\sum_{n=1}^{T} \log\left(\frac{|D_{n+1}|}{|D_n|}\right) \tag{1.4}$$

Jensen's inequality is used to estimate equation (1.4) with equation (1.5) [26]. The inequality states that equations (1.4) and (1.5) are equal if, $|D_1|/|D_0| = |D_2|/|D_1| = \cdots = |D_T|/|D_{T-1}|$, else (1.4) $<$ (1.5), as $\log(x)$ is a concave function.

$$\lambda(T) \;=\; \log\left(\frac{1}{T}\sum_{n=1}^{T} \frac{|D_{n+1}|}{|D_n|}\right) \tag{1.5}$$

The normalized Hamming size of a Boolean matrix is [15],

$$|\Omega| = \frac{1}{N^2}\sum_{i=0}^{N-1}\left(\sum_{j=0}^{N-1}\Omega_{i,j}\right)$$

$\Omega(n)$ is generated so it has an average of $k/2$ ones per row, thus, $|\Omega(t)| = k/2N$. The normalized Hamming size is equivalent to the probability of each element of the matrix (or vector) being 1. Use this and calculate, $|\Omega(n)D_n|$ using normal matrix multiplication.

$$\begin{aligned}\left|\begin{pmatrix} k/2N & \cdots & \\ \vdots & & \\ & & k/2N \end{pmatrix}\begin{pmatrix} |D_n| \\ \vdots \end{pmatrix}\right| &= \left|\begin{pmatrix} \frac{k}{2}|D_n| \\ \vdots \end{pmatrix}\right| \\ &= \frac{k}{2}|D_n| \\ &= N|\Omega(n)||D_n|\end{aligned}$$

Returning to equation (1.5),

$$\begin{aligned}\lambda(T) &= \log\left(\frac{1}{T}\sum_{n=1}^{T}\frac{|D_{n+1}|}{|D_n|}\right) \\ &= \log\left(\frac{1}{T}\sum_{n=1}^{T}\frac{N|\Omega(n)||D_n|}{|D_n|}\right) \\ &= \log\left(\frac{1}{T}\sum_{n=1}^{T}\frac{k}{2}\right) \\ &= \log\left(\frac{k}{2}\right) \qquad (1.6)\end{aligned}$$

Equation (1.6) agrees with Derrida and Pomeau: at $k = 2$ the dynamics of a classic RBN undergo a transition from stable ($k < 2$) to unstable ($k > 2$).

The final derivation of a measure for classic RBN dynamics, to be discussed here, is the stable core, s_n, Flyvbjerg [7]. s_n the proportion of nodes whose value has become fixed and will remain fixed for all time after n, independent of initial conditions. Flyvbjerg provides an analytical result for the transition in dynamical behaviour that agrees with Derrida and Pomeau, Luque and Solé, but does not use the annealed approximation (mean field approach). Flyvbjerg derives a probabilistic equation to describe the size evolution of the stable core,

$$s_{n+1} = \sum_{j=0}^{k}\binom{k}{j}s_n^{k-j}(1-s_n)^j p_j \qquad (1.7)$$

j is the number of variables a particular function has outside the stable core, p_j is the probability that the Boolean function is independent of those variables, $p_0 = 1$.

Flyvbjerg looks at the case where the stable core becomes fixed, $s_{n+1} = s_n = s$. Equation (1.7) with $s_{n+1} = s_n = s$ is always solved by $s = 1$ (which corresponds to all the network being frozen). Differentiating the right hand side of (1.7) with respect to s_n, we see that $s = 1$ is a fixed point of the iteration defined by equation (1.7) as long as $p_1 > 1 - \frac{1}{k}$.

Flyvbjerg refers to $p_1 = 1 - \frac{1}{k}$ as the critical condition for the network. Recall, p_j is the probability of a Boolean function having all but j variables inside the stable core and those j variables being ineffective. If there is only one variable outside the stable core then the Boolean function is effectively reduced to a Boolean function with one input. Two of the four Boolean functions with one variable are independent of that variable, f_i^0 and f_i^3, see table 1.4. As the Boolean functions are chosen at random (when defining F) we expect to get a Boolean function equivalent to one of these functions with probability $1/2$. Thus, $p_1 = 1/2$, and by the critical condition, $k = 2$. Once again we have a result stating that if a classic RBN is to show stable dynamics ($s = 1$) then $k = 2$.

$x_{i,l}$	f_i^0	f_i^1	f_i^2	f_i^3
0	0	1	0	1
1	0	0	1	1

Table 1.4: The truth table for all Boolean functions with one input

1.2 Probabilistic RBN's

In the previous section we talked about the dynamics of deterministic RBN's with classic and general topologies. There have been extensive studies into deterministic RBN's with more structured topologies, such as a mesh, ring, scale free, and evolving topologies [1] [20] [16], as well as the dynamical effects of certain groups of Boolean functions [24] [13]. We are interested in probabilistic (asynchronous) updating schemes, these were first considered in the context of genetic modelling as deterministic (synchronous) updating is thought to be a significantly incorrect assumption for gene behaviour [9]. In the deterministic setting dynamics are governed by a map F, in the probabilistic case maps are based on F but can change on each iteration. For example say we were considering the RBN described by map (1.1) with a probabilistic updating scheme such that only one

Boolean function (chosen randomly) is active on each iteration, the maps are,

$$\begin{array}{llll} x_0^{n+1} = f_0(x_1^n, x_0^n) & x_0^{n+1} = x_0^n & x_0^{n+1} = x_0^n & x_0^{n+1} = x_0^n \\ x_1^{n+1} = x_1^n & x_1^{n+1} = f_1(x_2^n, x_0^n) & x_1^{n+1} = x_1^n & x_1^{n+1} = x_1^n \\ x_2^{n+1} = x_2^n & x_2^{n+1} = x_2^n & x_2^{n+1} = f_2(x_1^n, x_3^n) & x_2^{n+1} = x_2^n \\ x_3^{n+1} = x_3^n & x_3^{n+1} = x_3^n & x_3^{n+1} = x_3^n & x_3^{n+1} = f_3(x_1^n, x_2^n) \end{array}$$

When thinking about probabilistic updating schemes it is simpler to talk in terms of networks and nodes. For the example above we say that one node is chosen at random to update on every iteration.

In 1997 Harvey and Bossomaier looked into the dynamics of asynchronous RBN's (ARBN's), their updating scheme is described above (each node has a mean update probability of $1/N$) [9]. They found that this updating scheme is a good method for searching out fixed points. Note that the fixed points of a deterministic RBN are preserved with the asynchronous updating scheme (and indeed any probabilistic updating scheme): for example let 0000 be a fixed point of a deterministic RBN with four nodes, i.e. $X^n = 0000 = X^{n+1}$ for all n, it is clear that if node 0, 1, 2, 3 (or any combination of nodes) were to update the network would still be in state 0000. The speed at which ARBN's found fixed points in Harvey and Bossomaiers work drew their attention to the fact that the fixed points of ARBN's have larger basins of attraction than their deterministic equivalents.

In order to get an idea of how often they should expect an ARBN to have a fixed point Harvey and Bossomaier perform a small calculation: they consider all possible networks with fixed N, k, and so can make the approximation that when a node is updated its value remains the same with probability $1/2$ (an annealed approximation). Thus the probability of being in a fixed point is $1/2^N$ (the probability that the state of the network remains the same after all nodes are updated). As $1/2^N$ is the probability of any of the 2^N states being a fixed point we can expect 1 fixed point per network.

One fixed point per network is only an approximation and many networks have no fixed points. Harvey and Bossomaier observed the dynamics of ARBN's which did not have fixed points fall into (what they define as) loose attractors. A loose attractor is a set of states that captures the trajectory of an ARBN. Di Paolo found that some trajectories in loose attractors had rhythmic properties (he used the autocorrelation between states to determine periods) [19].

Gershenson performed numerous simulations to tackle the concept of dynami-

cal stability in classic RBN's for a number of probabilistic updating schemes [8]. Among the schemes is the asynchronous scheme of Harvey and Bossomaier and a semi-synchronous scheme where a number of nodes are chosen at random on each iteration and updated synchronously. Gershenson decided to characterise stability using distance evolution (normalized Hamming distance). He chose two states distance $d_0 = 1/N$ apart, let them both run on the same network, subjecting both trajectories to the same random updating. After 10000 iterations Gershenson compared the distance between the two new states with the initial distance; he called this $\delta = d_{10000} - d_0, \quad -1/N \leq \delta \leq (N-1)/N$. These experiments were repeated a number of times and an average δ was found for various N, k and updating schemes.

Gershenson drew curves for δ against k (figure 1.6). He found that for $N = 100$ and 200, δ k curves are practically the same for all probabilistic updating schemes. In fact they follow the same trend as the deterministic δ k curve but the probabilistic δ is slightly larger. For $k = 2$ probabilistic $\delta \approx 0.08$, then increases rapidly to 0.5 as k increases (reaching 0.5 around $k = 4, 5$). Even though for $k = 2$ probabilistic δ is positive (indicating an increase in distance) it is small. That and the similarity between the probabilistic and deterministic δ k curves gives an indication that probabilistic updating schemes do not throw network dynamics into wildly unstable behaviours.

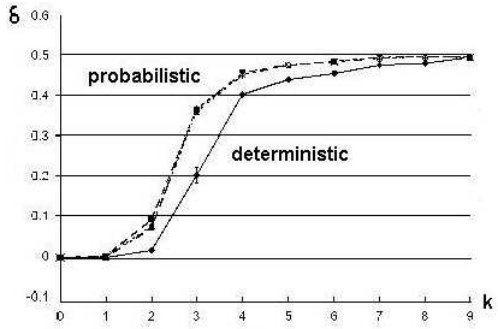

Figure 1.6: δ k curves for $N = 200$ classic RBN's, from Gershenson [8].

We will discuss the dynamics of probabilistic RBN's in this thesis, providing further numerical investigation into their dynamics in subsection 5.1.2, and proposing an annealed approximation for the transition between 'stable' and 'unstable' dynamics in subsection 5.1.3.

Chapter 2

Accuracy Boolean Networks

2.1 Accuracy heuristic motivation

This work germinated from a discussion with Keith Briggs (British Telecom, Adastral Park, Martlesham) [1] regarding internet traffic volume. Briggs talked about an idea for reducing traffic which bypasses routing/queueing protocols and topological issues, a method that could just sit in among existing software. The central idea is to trade accuracy of information for traffic.

A possible application for such an approach is the domain name system (DNS), particularly DNS replication (or redundancy). The DNS is a distributed database used to map machine names to IP addresses. To prevent the loss of any data and to improve performance, each DNS server is replicated at least twice [4]. The replicated servers contact each other periodically, once or twice a day, to check their files are up to date [4]. There were estimated to be over 2 million DNS servers in 2002 [18]. It may be worth noting that the traffic generated by replication maintenance (approximately 4 million messages a day in 2002) is small when we consider that some of the DNS root servers have reported 100 million queries per day [27]. (A root server is a main server which deals with and passes on queries regarding a particular zone. There are about 13 zones in all, an example of one of these is '.com'.) It was predicted that by 2005 the DNS would be 10 times larger than it was in 2002, which is an indication of the rate of increase in internet use [18]. We shall join the school of thought: if simple measures (no matter how small) can be taken to preserve internet resources they should be employed.

The DNS updating system is 'unintelligent', that is to say it uses fixed peri-

[1] `http://keithbiggs.info`

odic updating to maintain its databases. If the DNS were to employ a traffic minimizing updating scheme (in the same vein as the accuracy heuristic we will go on to describe) each server would receive information on request, and the decision on when to request would be based on the amount of changed data in the previously updated files, i.e. the more changed data the sooner the next update.

More generally, the networks we are thinking about could be described as 'condition satisfaction' networks. We have a collection of entries (the nodes of the network) each of which has control over one or more quantities; the nodes try to arrange their quantities (or 'states') to satisfy some conditions - but the conditions involve the values of quantities from other nodes. (Gene regulatory or other chemical networks are of this type.) Each node must communicate with other nodes to find out what values their quantities currently take, and then adjusts its own values to try and satisfy conditions. The main point about networks in this thesis is that communication is thought to be expensive, and should be minimized if possible. So the nodes try to minimize this traffic, but without having global information about the rest of the network.

2.2 Accuracy heuristic model

The simplest possible representation of a computer network would be a graph in which vertices represent individual computers and edges represent physical connections. (Generating such graphs with topologies that correctly model those of real computer networks, such as the internet, is a current area of research.) We can also think about models of more abstract 'information networks'. Here each vertex represents a store or collection of information (it may or may not be a single computer) and the edges represent the possibilities for passing information between the stores, though how this happens is not specified. The DNS is a computer network that is also an information network.

To account for what goes on within the nodes of the network some information must be specified for each node, that is the 'state' of that node. Clearly the internal state of a computer in a network can be very complicated, but in some situations the essential information might be represented more compactly, perhaps by just a few numbers. To make the most extreme simplification possible, the state could be a single number, which takes only two different values i.e. a Boolean variable. So the simplest possible model of an information network that we can think of is a Boolean network.

When thinking of using a Boolean network to model a distributed data base with redundancy, the change in state of a Boolean node could represent an alteration of the data stored in the processor being represented by the node. The dynamics of boolean networks can be manipulated using network topology or the Boolean function types (see chapter 1), so it is feasible that one could design a Boolean network to encompass both topological and behavioural features of a real network.

Regarding information traffic, we can say that re-evaluating a Boolean function generates traffic (as input states have to travel from predecessor nodes). This makes the deterministic RBN expensive. To reduce traffic costs a Boolean network could adopt a probabilistic updating scheme. So, not every node re-evaluates every time. Frequent updating leads to high accuracy (or good satisfaction of conditions) but also to high traffic; less frequent updating leads to lower accuracy but also lower traffic. The accuracy heuristic imposes probabilistic updating with a control parameter allowing the node to specify a minimum 'accuracy' of information (the term accuracy is explained in full in the next chapter). The heuristic itself is a stochastic equation employed by the node, that uses the node's output states to adjust state dependent re-evaluation probabilities, with the aim of achieving the required accuracy.

Chapter 3

Accuracy Heuristic

3.1 Accuracy heuristic

In order to discuss the accuracy heuristic we need only consider one node and its predecessors.

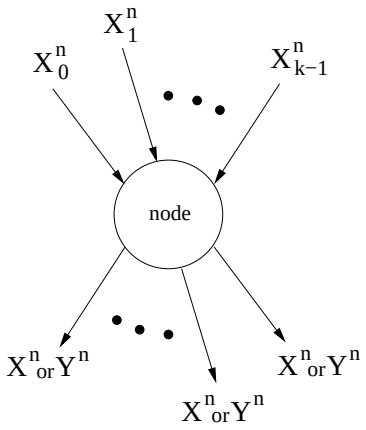

Figure 3.1: Diagram of a node, its inputs and output.

We are going to label two bit strings associated with the states of our node: At each time n, X^n is the state the node would adopt if there were an update at time n; Y^n is the state the node actually adopts. (Thus if there is an update $Y^n = X^n$ necessarily; if there is no update Y^n may or may not equal X^n.) Assume $\{X^n\}$ is independent and identically distributed (iid).

When the node updates it uses the states of its predecessors at time n, $X_0^n, X_1^n, \ldots, X_{k-1}^n$ and its Boolean function f, to determine its state at $n+1$, X^{n+1}, $(X^n, X_i^n \in \mathbf{B} = \{0,1\})$.

$$X^{n+1} \;=\; f(X_0^n, X_1^n, \ldots, X_{k-1}^n) \tag{3.1}$$

We will call equation (3.1) the accuracy condition. The $\{Y^n\}$ string is the stochastic process,

$$Y^{n+1} = \begin{cases} X^{n+1} & \text{with probability} \quad \tau_{y^n} \\ Y^n & \text{with probability} \quad 1-\tau_{y^n} \end{cases} \tag{3.2}$$

The accuracy heuristic provides a method for the node to compute τ_0 and τ_1, based upon (estimates of) the distributions of $\{X_n\}$ and $\{Y_n\}$. Since $\{X_n\}$ is assumed to be iid the values of τ_0 and τ_1 are taken to be independent of time. The idea of the heuristic is to chose τ_0, τ_1 so that the node state satisfies the accuracy condition with probability $1-\epsilon$ (where ϵ qualifies the extent to which we allow inaccuracy). We choose ϵ in the range $0 \leq \epsilon < 0.5$, where $\epsilon = 0$ means zero probability of inaccuracy, (and so $\tau_0 = \tau_1 = 1$). There are several ways of choosing τ_0, τ_1 such that $P(Y^{n+1} = X^{n+1}) = 1-\epsilon$, so we are going to choose a particular one.

Since the probability of update is state dependent it is convenient to express the probability of accuracy as,

$$\begin{aligned} P(Y^{n+1} = X^{n+1}) \;\; = \;\; & P(Y^{n+1} = X^{n+1} | Y^n = 0) P(Y^n = 0) + \\ & + P(Y^{n+1} = X^{n+1} | Y^n = 1) P(Y^n = 1) \end{aligned} \tag{3.3}$$

$Y^{n+1} = X^{n+1}$ holds either if the node updates, or if it does not and $Y^n = X^{n+1}$. So $P(Y^{n+1} = X^{n+1} | Y^n = y^n) = \tau_{y^n} + (1-\tau_{y^n}) P(X^{n+1} = y^n)$. The probability $P(X^{n+1} = y^n)$ is just the iid distribution on $\{X^n\}$, let $P(X^n = 1) = G_1$ and $P(X^n = 0) = G_0$. Also setting $P(Y^{n+1} = X^{n+1}) = 1-\epsilon$, equation (3.3) becomes,

$$1 - \epsilon = (\tau_0 + (1-\tau_0) G_0) P(Y^n = 0) + (\tau_1 + (1-\tau_1) G_1) P(Y^n = 1) \tag{3.4}$$

Equation (3.4) is the raw version of the accuracy heuristic. One way to ensure (3.4) is satisfied is to set $\tau_{y^n} + (1-\tau_{y^n}) G_{y^n} = 1-\epsilon$ for both $y^n = 0$ and $y^n = 1$ (then the right hand side of (3.4) becomes $(1-\epsilon) P(Y^n = 0) + (1-\epsilon) P(Y^n = 1) = (1-\epsilon)[P(Y^n = 0) + P(Y^n = 1)] = 1-\epsilon$).

To find another way of satisfying (3.4) note that we can rewrite it (using $H_0 = P(Y^n = 0)$, $H_1 = P(Y^n = 1)$) in the following way:

$$\begin{aligned} 1 - \epsilon \;\; &= \;\; 1 - (H_0 + H_1) + (\tau_0 + (1-\tau_0) G_0) H_0 + (\tau_1 + (1-\tau_1) G_1) H_1 \\ &= \;\; 1 - H_0 (1 - \tau_0 - (1-\tau_0) G_0) + H_1 (1 - \tau_1 - (1-\tau_1) G_1) \\ &= \;\; 1 - H_0 (1-\tau_0)(1-G_0) + H_1 (1-\tau_1)(1-G_1) \\ &= \;\; 1 - H_0 (1-\tau_0) G_1 + H_1 (1-\tau_1) G_0 \end{aligned}$$

so it is sufficient to set $H_{y^n}(1-\tau_{y^n})=\epsilon$ for $y^n=0,1$ since then the right hand side is $1-\epsilon G_1-\epsilon G_0=1-\epsilon$. This leads to,

$$\tau_{y^n}=1-\frac{\epsilon}{H_{y^n}} \tag{3.5}$$

This might at first seem an attractive expression for τ_{y^n}, since it involves only the quantity H_{y^n} which is directly accessible to the node. However from another point of view equation (3.5) is less desirable since it leads to more updating than the other expression ($\tau_{y^n}+(1-\tau_{y^n})G_{y^n}=1-\epsilon$). To see this observe that, however τ_0 and τ_1 are assigned, the probability that an update occurs at time $n+1$ is,

$$\begin{aligned}P(\text{update at } n+1|Y^n=0)P(Y^n=0)+P(\text{update at } n+1|Y^n=1)P(Y^n=1)\\ =\tau_0H_0+\tau_1H_1\end{aligned}$$

If τ_0, τ_1 are updated from (3.5), this probability is always $1-2\epsilon$. So now suppose τ_0, τ_1 are found from the other expression. For a given value of G_0 the update probabilities are fixed by $\tau_b=1-\epsilon/G_{\bar{b}}$ ($b\in\{0,1\}$, $\bar{b}=$ NOT b), and these probabilities together with G_0 (and G_1) determine H_0 (and H_1). Thus there is a relationship between H_0 and G_0; this is shown in the next section to be $H_0=(G_0-\epsilon)/(1-2\epsilon)$ (similarly $H_1=(G_1-\epsilon)/(1-2\epsilon)$). Hence the update probability in this case is,

$$\begin{aligned}\tau_0H_0+\tau_1H_1 &= \left(1-\frac{\epsilon}{G_1}\right)\left(\frac{G_0-\epsilon}{1-2\epsilon}\right)+\left(1-\frac{\epsilon}{G_0}\right)\left(\frac{G_1-\epsilon}{1-2\epsilon}\right)\\ &= \frac{(G_0-\epsilon)(G_1-\epsilon)}{(1-2\epsilon)G_0G_1}\\ &= \frac{1}{1-2\epsilon}\left(1-\frac{\epsilon(1-\epsilon)}{G_0G_1}\right)\end{aligned}$$

This is clearly a maximum when $G_0G_1=G_0(1-G_0)$ is a maximum, i.e. when $G_0=1/2$, in which case the update probability is $[1-4\epsilon(1-\epsilon)]/(1-2\epsilon)=1-2\epsilon$. Hence this method leads to as large an update probability as (3.5) only in the maximum case.

And so, we will use the expression for τ_{y^n} that satisfies equation (3.4),

$$\tau_{y^n}=1-\frac{\epsilon}{1-G_{y^n}}$$

This expression is only meaningful for $G_{y^n}\leq 1-\epsilon$; for $G_{y^n}>1-\epsilon$ it would imply $\tau_{y^n}<0$. So we still need to assign τ_{y^n} in this case. If the node is in state y^n and $G_{y^n}>1-\epsilon$ then if the node never updated it would be accurate with probability

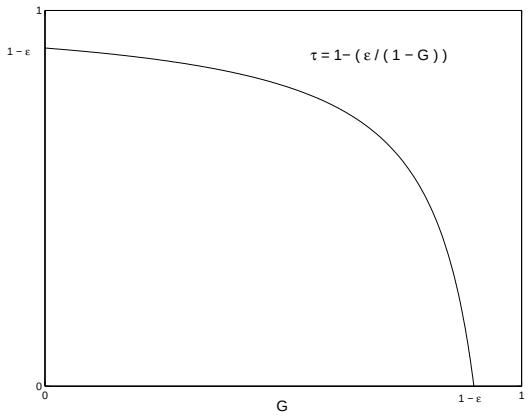

Figure 3.2: Graph of $\tau = 1 - \frac{\epsilon}{1-G}$ for $0 \leq G \leq 1 - \epsilon$.

G_{y^n}, which is a greater accuracy than required. So, for $G_{y^n} > 1 - \epsilon$ it is sensible to set $\tau_{y^n} = 0$, giving,

$$\tau_{y^n} = \begin{cases} 1 - \frac{\epsilon}{1-G_{y^n}} & \text{for, } 0 \leq G_{y^n} \leq 1 - \epsilon \\ 0 & \text{otherwise} \end{cases} \tag{3.6}$$

Equation (3.6) is the accuracy heuristic without time dependence. When using this equation in simulations the stationarity assumption on G_{y^n} does not necessarily hold. For a simulation, a Boolean network (of some topology) is set up to model an information network. It seems sensible that the initial re-evaluation probabilities on the network be equal to 1, giving an accurate network. Then we can impose a required accuracy on the network as a whole, or on individual nodes. The nodes running the heuristic estimate G_{y^n} using a memory containing $\{Y^n\}$, or a subset of $\{Y^n\}$ (subsections 3.2.1, 3.2.2), and use it to find a new value for τ_{y^n} from (3.6). Once a node is updating probabilistically ($\tau_{y^n} \neq 1$) the distribution of its output will be changed. If the number of nodes in the network that convert to probabilistic updating is sufficiently small (and the network is large and well-ramified) the inputs to a probabilistic node will be unaffected and any further estimations of G_{y^n} (τ_{y^n}) should remain more or less constant. However, if enough nodes change their output distributions, the inputs to our probabilistic node will change, and a new computation of τ_{y^n} at this node will produce different re-evaluation probabilities. Thus the nodes may undergo a series of τ_{y^n} re-evaluations, changing the state distributions across the network. We can include this time dependence in equation (3.6):

$$\tau_{y^n}^{n+1} = \begin{cases} 1 - \frac{\epsilon}{1-G_{y^n}^n} & \text{for, } 0 \leq G_{y^n}^n \leq 1 - \epsilon \\ 0 & \text{otherwise} \end{cases} \tag{3.7}$$

Equation (3.7) is used like so: at time n the node is in state y^n. Estimate $G^n_{y^n}$ using $\{Y^n\}$, (or a subset of $\{Y^n\}$, which may not include the value y_n). Equation (3.7) is used to give $\tau^{n+1}_{y^n}$, which is the re-evaluation probability used to generate y^{n+1}. Note, that it is the current state y^n which determines the re-evaluation probability to be used when generating the next state y^{n+1}, hence the notation $\tau^{n+1}_{y^n}$. We may not re-evaluate the τ_{y^n}'s every time. (For more thorough descriptions of network simulations see the Appendix.)

Dynamical studies and simulations strongly suggest that the values of $G^n_{y^n}$ ($\tau^n_{y^n}$) converge (sections 4.1, 4.2, 5.1). Nodes which converge with re-evaluation probabilities greater than zero always have a small probability of freezing due to the nature of probabilistic updating.

3.2 Estimating G

3.2.1 Methods for estimating G distribution

The accuracy heuristic (equation (3.7)) relies heavily on the value of G_b. Recall that G_b is the distribution of the process $\{X^n\}$ (section 3.1). Once a node is updating probabilistically it has no direct access to $\{X^n\}$, since we assume the node only obtains information about states of its predecessors if it actually updates. The node does have access to $\{Y^n\}$, the sequence of states it actually adopts. In this subsection we are going to introduce two methods of estimating G_b from the stochastic process $\{Y^n\}$ (equation (3.2)).

Note that $\{Y^n\}$ satisfies the Markov property [3],

$$P(Y^n = y^n | Y^{n-1} = y^{n-1}, Y^{n-2} = y^{n-2}, \ldots, Y^0 = y^0) = P(Y^n = y^n | Y^{n-1} = y^{n-1})$$

Let T be the transition matrix, $T_{i,j} = P(Y^n = i | Y^{n-1} = j)$. The event $(Y^n = 0, Y^{n-1} = 0)$ happens if, the node does not update or if it does update and $X^n = 0$. Thus,

$$\begin{aligned} P(Y^n = 0 | Y^{n-1} = 0) &= (1 - \tau_0) + \tau_0 G_0 \\ &= (1 - \tau_0) + \tau_0 (1 - G_1) \\ &= 1 - \tau_0 G_1 \end{aligned}$$

$(Y^n = 0, Y^{n-1} = 1)$ can only happen if the node updates and $X^n = 0$, giving $P(Y^n = 0 | Y^{n-1} = 1) = \tau_0 G_1$. The arguments for the other two events in the

transition matrix are very similar so,

$$T = \begin{pmatrix} 1-\tau_0 G_1 & \tau_1 G_0 \\ \tau_0 G_1 & 1-\tau_1 G_0 \end{pmatrix}$$

The stationary distribution $P(Y^n = 0)$, $P(Y^n = 1)$ is the invariant distribution for the Markov process, so to find it we need to find the eigenvector of T belonging to the unit interval. Letting $H_0 = P(Y_n = 0)$, $H_1 = P(Y_n = 1)$ we find,

$$H_0 = (1-\tau_0 G_1)H_0 + \tau_1 G_0 H_1$$

Rearrange to get G_b as a function of H_b, $H_{\bar{b}}$, τ_b, and $\tau_{\bar{b}}$,

$$G_b = \frac{\tau_b H_b}{\tau_b H_b + \tau_{\bar{b}} H_{\bar{b}}} \tag{3.8}$$

To learn a little about equation (3.8) we are going to substitute for τ_b using $\tau_b = 1 - \epsilon/(1-G_b)$,

$$G_b = H_b(1-2\epsilon) + \epsilon \tag{3.9}$$

$H_b \in [0,1]$ so $G_b \in [\epsilon, 1-\epsilon]$ and $\tau_b \in [0, (1-2\epsilon)/(1-\epsilon)]$. Equation (3.9) gives values for G_b which are within the interval $[0, 1-\epsilon]$ (see equation 3.6).

The first method for estimating G_b is to find H_b by observing the node state, and then to compute G_b using (3.8). Note however that this relationship may not be valid for a node in a network because some of the assumptions used to derive it may not be fulfilled, (i.e. $\{X^n\}$ being iid).

When using equation (3.8) in a simulation the time dependence is,

$$G_b^n = \frac{\tau_b^n H_b^n}{\tau_b^n H_b^n + \tau_{\bar{b}}^n H_{\bar{b}}^n}$$

To use the H G relationship (equation (3.8)) in a simulation, one has to calculate both τ_0^n and τ_1^n in order to get the estimate for G_b^n. We will also see that the variance of this G_b estimator is greater than the variance when estimating G_b directly (subsection 3.2.3). We will now talk about estimating G_b directly.

The second method for estimating G_b is to look at the state of the node when it updates. Although the node always knows Y^n, this may or may not be the same as X^n. However, if an update occurs at n (which the node can be assumed to know)

then certainly $Y^n = X^n$. Suppose $n_1, n_2, n_3, \ldots$ is the sequence of times at which updates occur, then the sequence $Y^{n_1}, Y^{n_2}, \ldots$ is identical to $X^{n_1}, X^{n_2}, \ldots$; and it is known by the node. An approach to estimating G_b is therefore to estimate the distribution of $X^{n_1}, X^{n_2}, \ldots$. Of course $\{X^{n_i}\}$ is not a random sampling of $\{X^n\}$ in the sense that n_{i+1} is strongly influenced by the value of X^{n_i}, (hence the update probabilities). However, as we shall see in subsection 3.2.3 we can estimate G_b from $\{X^{n_i}\}$ so long as we treat the data in the right way. Note that this again relies on $\{X^n\}$ being iid, and the estimate may not be accurate if this is false.

Estimating G_b directly involves a lot less computation than the first method and it is subject to less error. The only draw back may be that we have to wait longer than n to obtain n data points, drawing past events forward. This may delay convergence of G_b^n.

3.2.2 Memory types for estimating G distribution

The memory of a node is used to estimate a distribution by observing bits. In subsection 3.2.1 we discussed the nature of those bits, now we are going to describe what we mean by memory, and describe its various types.

We will say that a node has a vector memory, length N, if it fills a $N \times 1$ vector with bits, b_n, then performs the calculation,

$$P(b_n = 1) = \frac{1}{N} \sum_{n=1}^{N} b_n \tag{3.10}$$

Considered here are two ways to use the vector memory. One is emptying the vector after $P(b_n = 1)$ has been calculated. In this case the current value of $P(b_n = 1)$ is used until the vector is refilled and a more recent $P(b_n = 1)$ can be found. We call this memory an empty/fill vector memory.

During a simulation that uses an empty/fill vector memory, an actual vector is not necessary, the iteration s_n can be used for $n = 1, 2, \ldots, N$,

$$\begin{aligned} s_1 &= b_1 \\ s_n &= s_{n-1}\frac{n-1}{n} + \frac{b_n}{n} \end{aligned}$$

The other method we will term, the sliding vector method. The sliding vector is filled and equation (3.10) is used to find $P(b_n = 1)$. However, once this value is calculated, at each subsequent time step, the past most element of the vector is disregarded and a new element is introduced, then $P(b_n = 1)$ is recalculated.

The sliding vector gives up-to-date estimations of the distribution being calculated, but the cost is great. When simulating this vector we must keep track of all the bits (i.e. have an actual vector within the simulated node) which increases computing time, especially as the larger the value of N the better the estimate of $P(b_n = 1)$, see subsection 3.2.3. Also, it is our desire to have every node, within large networks, run the accuracy heuristic, rendering the sliding vector impractical.

To have an up-to-date estimate of $P(b_n = 1)$ is obviously desirable. We will now introduce the scalar memory.

As each update of the scalar memory uses its past values we shall denote $P(b_n = 1)$ at iteration n as P_n. The scalar memory uses an exponentially decaying average [10],

$$\begin{aligned} \varsigma_n &= \sum_{l=0}^{\infty} w^{l+1} b_{n-l} \\ &= \omega b_n + \omega^2 b_{n-1} + \omega^3 b_{n-2} + \cdots \end{aligned}$$

Clearly, ς_n is influenced by all past values of b_n. The parameter, $\omega \in (0, 1)$, is used to continually reduce the influence of each bit as its occurrence moves further into the past. To make an estimator for P_n we normalize ς_n to the range $[0, 1]$; that is, we say $P_n = \lambda\varsigma_n$ for some constant λ. To find a suitable λ, note $P_n = \lambda(\omega b_n + \omega^2 b_{n-1} + \cdots)$, so assuming the sequence $\{b_n\}$ is stationary we find (taking expectations) $P_n = P_n\lambda(\omega + \omega^2 + \cdots) = \frac{\omega}{1-\omega}\lambda P_n$. So $\lambda = \frac{1-\omega}{\omega}$, giving,

$$P_n = \varsigma_n \frac{1-\omega}{\omega}$$

In simulations we use the iteration,

$$P_n = (1-\omega)b_n + \omega P_{n-1}$$

from which it is clear that P_n is given by $(1-\omega)b_n + \omega(1-\omega)b_{n-1} + \cdots$. This iteration avoids storing past b_n's. It also makes clear that the latest P_n is a

weighted average of b_n with the previous estimate P_{n-1}, the larger ω the greater the influence of past events.

3.2.3 Memory Variance

This subsection is arranged to look firstly at the variance when estimating H_1 using the vector memory and the scalar memory, then the variance when estimating G_1 using both memories. We will derive general variance equations for H_1 and G_1, which we will then specialize to suit the vector and scalar memories. The data is generated with the state dependent sampling probabilities τ_0, τ_1. The sampling probabilities will be constant throughout the analysis.

Variance when estimating H_1

Consider the stochastic process $\{Y_n\}$, defined as,

$$\text{if } Y_n = b, \quad Y_{n+1} = \begin{cases} X_{n+1} & \text{with probability} \quad \tau_b \\ Y_n & \text{with probability} \quad 1-\tau_b \end{cases}$$

where $\{X_n\}$ is iid, and is modelling the output of a node within a synchronously updating network. The stochastic process, $\{Y_n\}$, represents the output of the same node updating with state dependent probabilities. Denote the estimate of H_1, on iteration n, as $\hat{\mu}_{Y_n}$: we will calculate the variance of $\hat{\mu}_{Y_n}$. The distribution on $\{Y_n\}$ is stationary as it is generated by sampling an iid stochastic process, using fixed sampling probabilities, and so $H_1 = \langle Y_n \rangle = \mu_Y$ is constant.

The simplest form of estimator is a weighted sum over Y_n,

$$\hat{\mu}_{Y_n} = a_0 Y_n + a_1 Y_{n-1} + \cdots + a_{N-1} Y_{n-(N-1)} \tag{3.11}$$

where N is the number of terms in the sum, and controls how far back into the past we look when estimating $\hat{\mu}_{Y_n}$. Clearly,

$$\begin{aligned} \langle \hat{\mu}_{Y_n} \rangle &= \sum_{i=0}^{N-1} a_i \langle Y_{n-i} \rangle \\ &= \mu_Y \sum_{i=0}^{N-1} a_i \end{aligned}$$

Thus for an unbiased estimator we need to choose the a_i's so that $\sum_{i=0}^{N-1} a_i = 1$. With that choice, the variance of $\hat{\mu}_{Y_n}$ is $\langle (\hat{\mu}_{Y_n} - \mu_Y)^2 \rangle$. Letting $(Y_{n-i} - \mu_Y) = \psi_{n-i}$,

we have,

$$\begin{aligned}(\hat{\mu}_{Y_n} - \mu_Y)^2 &= \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} a_i a_j \psi_{n-i}\psi_{n-j} \\ \left\langle(\hat{\mu}_{Y_n} - \mu_Y)^2\right\rangle &= \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} a_i a_j \left\langle\psi_{n-i}\psi_{n-j}\right\rangle \end{aligned} \tag{3.12}$$

The expectation, $\langle\psi_{n-i}\psi_{n-j}\rangle$, is the autocovariance of $\{Y_n\}$ at lag $m = j - i$, denoted $c(m)$. Re-indexing equation (3.12),

$$\begin{aligned}\left\langle(\hat{\mu}_{Y_n} - \mu_Y)^2\right\rangle &= \sum_{m=0}^{N-1}\sum_{i=0}^{N-1-m} a_i a_{i+m}\left\langle\psi_{n-i}\psi_{n-(i+m)}\right\rangle + \\ &\qquad + \sum_{m=-(N-1)}^{-1}\sum_{i=-m}^{N-1} a_i a_{i+m}\left\langle\psi_{n-i}\psi_{n-(i+m)}\right\rangle \\ &= \sum_{i=0}^{N-1} a_i^2\left\langle\psi_{n-i}^2\right\rangle + \sum_{m=1}^{N-1} c(m)\sum_{i=0}^{N-1-m} a_i a_{i+m} + \\ &\qquad + \sum_{m=-(N-1)}^{-1} c(m)\sum_{i=-m}^{N-1} a_i a_{i+m}\end{aligned}$$

Note that $c(m) = c(-m)$ because of stationarity. Also note that if $m' = -m$ we have $\sum_{i=0}^{N-1-m} a_i a_{i+m} = \sum_{i=-m'}^{N-1} a_i a_{i+m'}$, so the positive and negative m sums are identical. Using $\left\langle\psi_{n-i}^2\right\rangle = \sigma_Y^2$, the variance of Y_i:

$$\left\langle(\hat{\mu}_{Y_n} - \mu_Y)^2\right\rangle = \sigma_Y^2\sum_{i=0}^{N-1} a_i^2 + 2\sum_{m=1}^{N-1} c(m)\sum_{i=0}^{N-1-m} a_i a_{i+m} \tag{3.13}$$

Equation (3.13) is the variance for the general estimator, equation (3.11).

Variance when estimating H_1 using a vector memory

For vector estimation, a vector length N, is filled with the past N outputs of the node, namely $y_n, y_{n-1}, y_{n-2}, \ldots, y_{n-(N-1)}$. We then estimate H_1 by summing the elements and dividing by N. It is not relevant to variance calculations if the vector memory is a sliding vector or a empty/fill vector, (subsection 3.2.2). If it is sliding, on the next iteration the past most element is replaced by the next element of the stochastic process and H_1 is calculated again. Otherwise, the estimation for H_1 is used for the next N iterations before H_1 is recalculated. Thus, for the case where $\hat{\mu_{Y_n}}$ is being estimated by a vector, length N, $a_i = 1/N$,

$\forall\ i$, and equation (3.13) becomes,

$$\begin{aligned}\left\langle(\hat{\mu}_{Y_n}-\mu_Y)^2\right\rangle_v &= \sigma_Y^2\sum_{i=0}^{N-1}\frac{1}{N^2}+2\sum_{m=1}^{N-1}c(m)\sum_{i=0}^{N-1-m}\frac{1}{N^2}\\ &= \frac{\sigma_Y^2}{N}+\frac{2}{N^2}\sum_{m=1}^{N-1}(N-m)c(m) \qquad (3.14)\end{aligned}$$

Recall,

$$\begin{aligned}\sigma_Y^2 &= \left\langle\psi_{n-i}^2\right\rangle\\ &= \mu_Y(1-\mu_Y)\end{aligned}$$

The HG relationship (equation (3.8)) can be re-arranged to give μ_Y in terms of τ_0, τ_1 and μ_X (μ_X = the expectation of $\{X_n\}$). Let $\mu_X = p$.

$$\mu_Y = \frac{\tau_0 p}{\tau_0 p+\tau_1(1-p)}$$

and so,

$$\sigma_Y^2 = \frac{\tau_0\tau_1 p(1-p)}{(\tau_0 p+\tau_1(1-p))^2}$$

It is useful to let $A=\tau_0 p+\tau_1(1-p)$ when calculating $c(m)$. We want to know $c(m)$ as a function of τ_0 and τ_1.

$$c(m)=\langle Y_i Y_{i+m}\rangle-\mu_Y^2$$

In order to find an expression for $\langle Y_i Y_{i+m}\rangle$, we can use the fact that $\{Y_n\}$ is Markov and has transition matrix, T.

$$T=\begin{pmatrix}1-\tau_0 p & \tau_1(1-p)\\ \tau_0 p & 1-\tau_1(1-p)\end{pmatrix}$$

The element $T_{1,1}^m$ of T^m is the conditional probability $P(y_{i+m}=1|y_i=1)$, so $\langle Y_i Y_{i+m}\rangle = P(Y_{i+m}=1|Y_i=1)P(Y_i=1)$, and $c(m)=\mu_Y(T_{1,1}^m-\mu_Y)$.

Let, $\bigwedge = \text{diag}\{\lambda_1\lambda_2\}$ and $\bigvee = (v_1 v_2)$, where λ_i are the eigenvalues of T and v_i are the corresponding eigenvectors. We know that $\lambda_1=1$, so from the trace of T we find $\lambda_2 = 1-\tau_0 p-\tau_1(1-p)=1-A$. We solve $(T-\lambda I)\underline{v}=0$ to get eigenvectors, $v_1^T=(1,\tau_0 p/\tau_1(1-p))$ and $v_2^T=(1,-1)$. Now we are in a position to calculate T^m.

$$\begin{aligned}T^m &= \bigvee\bigwedge^m\bigvee^{-1}\\ &= -\tau_1(1-p)\frac{1}{A}\begin{pmatrix}1 & 1\\ \frac{\tau_0 p}{\tau_1(1-p)} & -1\end{pmatrix}\begin{pmatrix}1 & 0\\ 0 & (1-A)^m\end{pmatrix}\begin{pmatrix}-1 & -1\\ \frac{-\tau_0 p}{\tau_1(1-p)} & 1\end{pmatrix}\\ &= -\tau_1(1-p)\frac{1}{A}\begin{pmatrix}1 & (1-A)^m\\ \frac{\tau_0 p}{\tau_1(1-p)} & -(1-A)^m\end{pmatrix}\begin{pmatrix}-1 & -1\\ \frac{-\tau_0 p}{\tau_1(1-p)} & 1\end{pmatrix}\end{aligned}$$

From here we can deduce $T^m_{1,1}$,

$$\begin{aligned} T^m_{1,1} &= -\tau_1(1-p)\frac{1}{A}\left(\frac{-\tau_0 p}{\tau_1(1-p)} - (1-A)^m\right) \\ &= \frac{\tau_0 p}{A} + \tau_1(1-p)\frac{1}{A}(1-A)^m \end{aligned}$$

and $c(m)$,

$$\begin{aligned} c(m) &= \frac{\tau_0 p}{A}\left(\frac{\tau_0 p}{A} + \tau_1(1-p)\frac{1}{A}(1-A)^m\right) - \frac{(\tau_0 p)^2}{A^2} \\ &= \frac{\tau_0\tau_1 p(1-p)}{A^2}(1-A)^m \\ &= \sigma_Y^2 \lambda_2^m \end{aligned}$$

We are now in a position to write down the equation for the variance of the vector estimator. Equation (3.13) becomes,

$$\begin{aligned} \left\langle(\hat{\mu}_{Y_n} - \mu_Y)^2\right\rangle_v &= \frac{\sigma_Y^2}{N}\left(1 + \frac{2}{N}\sum_{m=1}^{N-1}(N-m)\lambda_2^m\right) \\ &= \frac{\tau_0\tau_1 p(1-p)}{NA^2}\left(1 + \frac{2}{N}\left(N\sum_{m=1}^{N-1}(1-A)^m - \sum_{m=1}^{N-1} m(1-A)^m\right)\right) \\ &= \frac{\tau_0\tau_1 p(1-p)}{NA^2}\left(1 + \frac{2}{N}\left(N\sum_{m=1}^{N-1}(1-A)^m - \sum_{m=1}^{N-1}\sum_{l=m}^{N-1}(1-A)^l\right)\right) \\ &= \frac{\tau_0\tau_1 p(1-p)}{NA^2}\left(1 + \frac{2}{N}\left(N\sum_{m=1}^{N-1}(1-A)^m + \right.\right. \\ &\qquad \left.\left. + \frac{1}{A}\left(\sum_{m=1}^{N-1}(1-A)^N - \sum_{m=1}^{N-1}(1-A)^m\right)\right)\right) \\ &= \frac{\tau_0\tau_1 p(1-p)}{NA^2}\left(1 + \frac{2}{NA}\left(\left(N - \frac{1}{A}\right)(1-A)(1-(1-A)^{N-1}) + \right.\right. \\ &\qquad \left.\left. + (N-1)(1-A)^N\right)\right) \end{aligned}$$

$$\left\langle(\hat{\mu}_{Y_n} - \mu_Y)^2\right\rangle_v = \frac{\tau_0\tau_1 p(1-p)}{NA^2}\left(1 + \frac{2}{NA}(1-A)\left(N + \frac{1}{A}((1-A)^N - 1)\right)\right)$$

$\left\langle(\hat{\mu}_{Y_n} - \mu_Y)^2\right\rangle_v$ is the variance when estimating μ_Y. We require the variance when estimating μ_X (G_1), which can be found using the HG relationship (equation (3.8)),

$$\mu_X = \frac{\tau_1\mu_Y}{\tau_0 + \mu_Y(\tau_1 - \tau_0)}$$

This relationship is not linear but as long as the variance is small we can use the equation,

$$\text{var } \mu_X = \left(\frac{d\mu_X}{d\mu_Y}\right)^2 \text{var } \mu_Y$$

Giving,

$$\text{var } \mu_X = \left(\frac{A}{\tau_0\tau_1}\right)^2 \left\langle (\hat{\mu}_{Y_n} - \mu_Y)^2 \right\rangle \tag{3.15}$$

And so for the vector memory we have,

$$(\text{var } \mu_X)_v = \frac{p(1-p)}{N\tau_0\tau_1}\left(1 + \frac{2}{NA}(1-A)\left(N + \frac{1}{A}((1-A)^N - 1)\right)\right) \tag{3.16}$$

Variance when estimating H_1 using a scalar memory

For the scalar memory, previous estimations of H_1 influence the current estimate of H_1: thus we will use the notation H_1^n. To estimate H_1^n the output of a node is weighted by $\omega^i \in (0,1)$, so that the past most output has the least effect,

$$H_1^n = \omega H_1^{n-1} + (1-\omega)y_n \tag{3.17}$$

When specializing equation (3.13) for the scalar estimator, N tends to infinity, and the a_i's are now different for each i. To determine a_i we look at the time evolution of the estimator, equation (3.17),

$$\begin{aligned}
\hat{\mu}_{Y_0} &= (1-\omega)Y_0 \\
\hat{\mu}_{Y_1} &= \omega(1-\omega)Y_0 + (1-\omega)Y_1 \\
\hat{\mu}_{Y_2} &= \omega^2(1-\omega)Y_0 + \omega(1-\omega)Y_1 + (1-\omega)Y_2 \\
\vdots &= \vdots \\
\hat{\mu}_{Y_n} &= \omega^n(1-\omega)Y_0 + \omega^{n-1}(1-\omega)Y_1 + \omega^{n-2}(1-\omega)Y_2 + \cdots + \omega^0(1-\omega)Y_n
\end{aligned}$$

so, $a_i = \omega^i(1-\omega)$. Putting this into equation (3.13) and letting $N \to \infty$ we get,

$$\begin{aligned}
\left\langle (\hat{\mu}_{Y_n} - \mu_Y)^2 \right\rangle_s &= (1-\omega)^2\sigma_Y^2 \sum_{i=0}^{\infty}\omega^{2i} + 2(1-\omega)^2 \sum_{m=1}^{\infty} c(m) \sum_{i=0}^{\infty}\omega^{m+2i} \\
&= \sigma_Y^2 \frac{(1-\omega)^2}{1-\omega^2} + 2\frac{(1-\omega)^2}{1-\omega^2}\sum_{m=1}^{\infty} c(m)\omega^m \\
&= \frac{1-\omega}{1+\omega}\left(\sigma_Y^2 + 2\sum_{m=1}^{\infty} c(m)\omega^m\right) \\
&= \sigma_Y^2 \frac{1-\omega}{1+\omega}\left(1 + 2\sum_{m=1}^{\infty}(\omega(1-A))^m\right)
\end{aligned} \tag{3.18}$$

σ_Y^2 and $c(m)$ are as for the vector estimator.

$\omega \in (0,1)$, so the infinite sum $\sum_{m=1}^{\infty}(\omega(1-A))^m$ converges if $-1 < (1-A) < 1$ (i.e. if $0 < A < 2$). In fact, $A = \tau_0 p + (1-p)\tau_1 \geq 0$, and $A = \tau_0 p + (1-p)\tau_1 \leq p\max\{\tau_0,\tau_1\} + (1-p)\max\{\tau_0,\tau_1\} = \max\{\tau_0,\tau_1\} \leq 1$. So, $0 \leq A \leq 1$, and the sum converges.

$$\begin{aligned}\left\langle(\hat{\mu}_{Y_n} - \mu_Y)^2\right\rangle_s &= \frac{\tau_0\tau_1 p(1-p)(1-\omega)}{A^2(1+\omega)}\left(1 + \frac{2\omega(1-A)}{1-\omega(1-A)}\right) \\ &= \frac{\tau_0\tau_1 p(1-p)(1-\omega)(1+\omega(1-A))}{A^2(1+\omega)(1-\omega(1-A))}\end{aligned}$$

Using equation (3.15) we get

$$(\text{var } \mu_X)_s = \frac{p(1-p)(1-\omega)(1+\omega(1-A))}{\tau_0\tau_1(1+\omega)(1-\omega(1-A))} \tag{3.19}$$

A special case of state dependent probabilistic updating is $\tau_0 = \tau_1 = \tau$, state independent probabilistic updating. Under this condition; $\mu_Y = \mu_X = p$, $\sigma_Y^2 = p(1-p)$, $A = \tau$, and,

$$\left\langle(\hat{\mu}_{Y_n} - \mu_Y)^2\right\rangle_v = \frac{p(1-p)}{N}\left(1 + \frac{2}{N\tau}(1-\tau)\left(N + \frac{1}{\tau}((1-\tau)^N - 1)\right)\right)$$

$$\left\langle(\hat{\mu}_{Y_n} - \mu_Y)^2\right\rangle_s = \frac{p(1-p)(1-\omega)(1+\omega(1-\tau))}{(1+\omega)(1-\omega(1-\tau))}$$

Variance when estimating G_1

To estimate G_1 directly we record Y_n (and include it in our estimation) only at times n at which an update actually occurs; we know that at such times $Y_n = X_n$. The Y_n's recorded therefore form a subsequence of the complete $\{Y_n\}$ sequence (and of the complete $\{X_n\}$ sequence). The indices of the Y_n's that we include in the subsequence are random variables depending on the update probabilities τ_0 and τ_1.

For each time n we say that the time of the latest update before, or at, n is $n - K_0$ ($K_0 = 0$ if the latest update is at n); the time of the update before

this is $n - K_0 - K_1$, and so on. $K_0, K_1, \ldots$ are integer random variables with $K_0 \geq 0$, and $K_i \geq 1$, $i > 0$. The state of the node is recorded when an update occurs, so at time n the recorded values are $Y_{n-K_0}, Y_{n-K_0-K_1}, \ldots$ (or equivalently $X_{n-K_0}, X_{n-K_0-K_1}, \ldots$).

To estimate G_1 we use a weighted sum over the recorded states:

$$\hat{\mu}_{X_n} = a_0 X_{n-K_0} + a_1 X_{n-(K_0+K_1)} + \cdots + a_{N-1} X_{n-(K_0+K_1+\cdots+K_{N-1})} \tag{3.20}$$

Note that expression (3.11) does not cover this case since the indices included in the sum are now random. So the first thing we need to investigate is $\langle \hat{\mu}_X \rangle$ (Is this estimator unbiased?).

$$\langle \hat{\mu}_{X_n} \rangle = a_0 \langle X_{n-K_0} \rangle + a_1 \left\langle X_{n-(K_0+K_1)} \right\rangle + \cdots + a_{N-1} \left\langle X_{n-(K_0+K_1+\cdots+K_{N-1})} \right\rangle$$

The expectation of X_{n-K_0} is $P(X_{n-K_0} = 1)$. The calculation of $P(X_{n-K_0} = 1)$ involves considering all possible values of K_0.

$$\begin{aligned} \langle X_{n-K_0} \rangle &= P(X_{n-K_0} = 1) \\ &= P\left(\bigcup_{k_0=0}^{\infty} \{X_{n-K_0} = 1, K_0 = k_0\} \right) \\ &= \sum_{k_0=0}^{\infty} P(X_{n-K_0} = 1, K_0 = k_0) \end{aligned} \tag{3.21}$$

The probability of $\{X_{n-K_0} = 1, K_0 = k_0\}$ is, the probability that an update took place $n - k_0$ iterations ago, yielding $X_{n-K_0} = 1$, and no updates took place thereafter. We are dealing with state dependent update probabilities, so the value of X_{n-K_0}, say x_0, affects the value k_0 in so much that it determines τ_{x_0}. However, once τ_{x_0} is set the two events, $X_{n-K_0} = x_0$ and $K_0 = k_0$, are essentially independent.

To compute $P(X_{n-K_0} = 1, K_0 = k_0)$ note that we can write the event $\{X_{n-K_0} = 1, K_0 = k_0\}$ as $\{\text{no update at } n, \text{no update at } n-1, \ldots, \text{no update at } n-k_0+1, \text{update at } n-k_0, X_{n-k_0} = 1\}$, so

$P(X_{n-K_0} = 1, K_0 = k_0) =$
$P(\text{no update at } n | \text{no update at } n-1, \ldots, \text{no update at } n-k_0+1, \text{update at } n-k_0, X_{n-k_0} = 1) \times P(\text{no update at } n-1, \ldots, \text{no update at } n-k_0+1, \text{update at } n-k_0, X_{n-k_0} = 1)$

Now the event {no update at $n-1, \ldots,$ no update at $n-k_0+1$, update at $n-k_0, X_{n-k_0}=1$} implies that $Y_{n-1}=1$, so the first factor in the above equation is

$$P(\text{no update at } n | Y_{n-1}=1) = 1-\tau_1$$

because once we know Y_{n-1} this fixes the update probability at time n, and the inclusion of any further information about what happened before $n-1$ does not alter the update probability. Thus

$P(X_{n-K_0}=1, K_0=k_0) = (1-\tau_1)P(\text{no update at } n-1, \ldots, \text{no update at } n-k_0+1, \text{update at } n-k_0, X_{n-k_0}=1)$

Applying the same reasoning k_0-1 more times gives

$$P(X_{n-K_0}=1, K_0=k_0) = (1-\tau_1)^{k_0} P(\text{update at } n-k_0, X_{n-k_0}=1)$$

Since the X sequence is iid, the probability of update at time $n-k_0$ is independent of X_{n-k_0} (it depends only on previous X_i's). So

$$\begin{aligned} P(\text{update at } n-k_0, X_{n-k_0}=1) &= P(\text{update at } n-k_0)P(X_{n-k_0}=1) \\ &= pP_u \end{aligned}$$

Where P_u is the probability that an update takes place at any given time, and hence at $n-k_0$. Thus,

$$P(X_{n-K_0}=1, K_0=k_0) = (1-\tau_1)^{k_0} pP_u$$

and equation (3.21) becomes,

$$\begin{aligned} \langle X_{n-K_0} \rangle &= P\left(\bigcup_{k_0=0}^{\infty} \{X_{n-K_0}=x_0, K_0=k_0\} \right) \\ &= pP_u \sum_{k_0=0}^{\infty} (1-\tau_1)^{k_0} \\ &= \frac{P_u}{\tau_1} p \end{aligned}$$

Calculating P_u: The probability of an update at any one time is either, τ_0 with probability $(1-\mu_Y)$, or τ_1 with probability μ_Y. Recall, $\mu_Y = \tau_0 p/(\tau_0 p + \tau_1(1-p))$. so,

$$\begin{aligned} P_u &= (1-\mu_Y)\tau_0 + \mu_Y \tau_1 \\ &= \frac{\tau_0 \tau_1}{\tau_0 p + \tau_1(1-p)} \end{aligned} \tag{3.22}$$

And so, the expectation of X_{n-K_0} is,

$$\begin{aligned}\langle X_{n-K_0}\rangle &= p\frac{P_u}{\tau_1}\\ &= \mu_Y\end{aligned}$$

It transpires that, $\left\langle X_{n-(K_0+\cdots+K_j)}\right\rangle = p \ \forall \ j \geq 1$. To show this we will use induction.

First consider $\left\langle X_{n-(K_0+K_1)}\right\rangle$.

$$\begin{aligned}\left\langle X_{n-(K_0+K_1)}\right\rangle &= P(X_{n-(K_0+K_1)}=1)\\ &= P\left(\bigcup_{x_0=0}^{1}\bigcup_{k_0=0}^{\infty}\bigcup_{k_1=1}^{\infty}\left\{X_{n-(K_0+K_1)}=1, X_{n-K_0}=x_0, K_0=k_0, K_1=k_1\right\}\right)\\ &= \sum_{x_0=0}^{1}\sum_{k_0=0}^{\infty}\sum_{k_1=1}^{\infty}P(X_{n-(K_0+K_1)}=1, X_{n-K_0}=x_0, K_0=k_0, K_1=k_1)\end{aligned}$$

To compute $P(X_{n-(K_0+K_1)}=1, X_{n-K_0}=x_0, K_1=k_1, K_0=k_0)$ we write the event as,

{no update at n, no update at $n-1,\ldots,$ no update at $n-k_0+1$, update at $n-k_0, X_{n-k_0}=x_0$, no update at $n-k_0-1$, no update at $n-k_0-2,\ldots,$ no update at $n-k_0-k_1-1$, update at $n-k_0-k_1, X_{n-(k_0+k_1)}=1$}

We break down the probability of this event in the same way as $P(X_{n-K_0}=1, K_0=k_0)$, but now we are looking at $P(X_{n-K_0}=x_0, K_0=k_0)$,

$$\begin{aligned}& P(X_{n-(K_0+K_1)}=1, X_{n-K_0}=x_0, K_1=k_1, K_0=k_0)\\ =\ & (1-\tau_{x_0})^{k_0}P(X_{n-k_0}=x_0, \text{update at } n-k_0,\ldots,\text{update at } n-k_0-k_1, X_{n-(k_0+k_1)}=1)\\ =\ & (1-\tau_{x_0})^{k_0}P(X_{n-k_0}=x_0)P(\text{update at } n-k_0,\ldots,\text{update at } n-k_0-k_1, X_{n-(k_0+k_1)}=1)\end{aligned} \tag{3.23}$$

(since X_{n-k_0} is independent of whether there is an update at $n-k_0$, $n-k_0-1$ etc, and is independent of $X_{n-k_0-k_1}$, since $\{X_n\}$ is iid). Noting that,

$P($update at $n-k_0,$ no update at $n-k_0-1,\ldots,$ update at $n-k_0-k_1, X_{n-k_0-k_1}=1)$
$P($update at $n-k_0|$ no update at $n-k_0-1,\ldots,$ update at $n-k_0-k_1, X_{n-k_0-k_1}=1)$
$\times P($ no update at $n-k_0-1,\ldots,$ update at $n-k_0-k_1, X_{n-k_0-k_1}=1)$

and that,
(no update at $n-k_0-1,\ldots,$ update at $n-k_0-k_1, X_{n-k_0-k_1}=1)$ implies $Y_{n-k_0-1}=1$,

we see that the third factor in equation (3.23) is,

$$\tau_1 P(\text{no update at } n-k_0-1,\ldots,\ \text{update at } n-k_0-k_1, X_{n-k_0-k_1}=1)$$

Hence,

$$\begin{aligned}
& P(X_{n-K_0-K_1}=1, X_{n-K_0}=x_0, K_1=k_1, K_0=k_0) \\
= \;& (1-\tau_{x_0})^{k_0} p^{x_0}(1-p)^{1-x_0}\tau_1 P(\text{ no update at } n-k_0-1,\ldots \\
& \qquad\qquad \ldots,\ \text{update at } n-k_0-k_1, X_{n-k_0-k_1}=1) \\
= \;& (1-\tau_{x_0})^{k_0} p^{x_0}(1-p)^{1-x_0}\tau_1(1-\tau_1)^{k_1-1}pP_u
\end{aligned}$$

Finally,

$$\begin{aligned}
\langle X_{n-K_0-K_1}\rangle &= \sum_{x_0=0}^{1}\sum_{k_0=0}^{\infty}\sum_{k_1=1}^{\infty} P(X_{n-K_0-K_1}=1, X_{n-K_0}=x_0, K_0=k_0, K_1=k_1) \\
&= \sum_{k_0=0}^{\infty}\sum_{k_1=1}^{\infty}\big[pP_u(1-p)\tau_1(1-\tau_0)^{k_0}(1-\tau_1)^{k_1-1}+ \\
&\qquad\qquad +p^2P_u\tau_1(1-\tau_1)^{k_0}(1-\tau_1)^{k_1-1}\big] \\
&= pP_u\left(\frac{(1-p)}{\tau_0}+\frac{p}{\tau_1}\right) \\
&= p
\end{aligned}$$

Let L_j be the sum of random variables $K_0+K_1+\cdots+K_j$ and $l_j = k_0+k_1+\cdots+k_j$. To show $\langle X_{n-L_j}\rangle = p\ \forall\ j\geq 1$, we are going to prove, by induction on i, that

$$\begin{aligned}
& P(K_0=k_0, X_{n-L_0}=x_0, K_1=k_1, X_{n-L_1}=x_1,\ldots,K_i=k_i, X_{n-L_i}=x_i) \\
& \quad = (1-\tau_{x_0})^{k_0}P(X_{n-l_0}=x_0)\tau_{x_1}(1-\tau_{x_1})^{k_1-1}P(X_{n-l_1}=x_1)\ldots \\
& \qquad\qquad \ldots\tau_{x_i}(1-\tau_{x_i})^{k_i-1}P(X_{n-l_i}=x_i)P_u \qquad (3.24)
\end{aligned}$$

Note, we have established this for $i=0$ above.

Write the event,

$$u_{i+1} = \{K_0=k_0, X_{n-L_0}=x_0, K_1=k_1, X_{n-L_1}=x_1,\ldots,K_{i+1}=k_{i+1}, X_{n-L_{i+1}}=x_{i+1}\}$$

as,

$\{$no update at $n,$ no update at $n-1,\ldots,$ no update at $n-l_0+1,$ update at $n-$

$l_0, X_{n-l_0} = x_0, \ldots, \text{update at } n-l_1, X_{n-l_1} = x_1, \ldots \text{update at } n-l_i, X_{n-l_i} = x_i\}$

Following the reasoning of previous calculations,

$$\begin{aligned}
P(u_{i+1}) &= P(\text{no update at } n | \text{ no update at } n-1, \ldots, X_{n-l_i} = x_i) \\
&\qquad \times P(\text{ no update at } n-1, \ldots, X_{n-l_i} = x_i) \\
&= (1-\tau_{x_0})^{k_0} P(X_{n-l_0} = x_0, \text{update at } n-k_0, \ldots, X_{n-l_i} = x_i) \\
&= (1-\tau_{x_0})^{k_0} P(X_{n-l_0} = x_0) P(\text{update at } n-l_0, \ldots, X_{n-l_i} = x_i) \\
&= (1-\tau_{x_0})^{k_0} P(X_{n-l_0} = x_0) \tau_{x_1} P(\text{no update at } n-l_0-1, \ldots, X_{n-l_i} = x_i) \\
&= (1-\tau_{x_0})^{k_0} P(X_{n-l_0} = x_0) \tau_{x_1} \left[(1-\tau_{x_1})^{k_1-1} P(X_{n-l_1} = x_1) \tau_{x_2} (1-\tau_{x_2})^{k_2-1} \right. \\
&\qquad \left. \times P(X_{n-l_2} = x_2) \ldots \tau_{x_{i+1}} (1-\tau_{x_{i+1}})^{k_{i+1}-1} P(X_{n-l_{i+1}} = x_{i+1}) P_u \right]
\end{aligned}$$

Where for the last step we use the induction hypothesis (equation (3.24)), replacing n by $n-l_0-1$, k_0 by k_1-1, l_1 by l_2, and x_0 by x_1, etc. Thus the hypothesis is true for i+1.

To find $\left\langle X_{n-L_j} \right\rangle$ we compute,

$$\begin{aligned}
P(X_{n-L_j} = 1) &= \sum_{k_0=0}^{\infty} \sum_{k_1=1}^{\infty} \cdots \sum_{k_j=1}^{\infty} \sum_{x_0=0}^{1} \cdots \sum_{x_{j-1}=0}^{1} P(K_0 = k_0, X_{n-L_0} = x_0, K_1 = k_1, \\
&\qquad X_{n-L_1} = x_1, \ldots, K_j = k_j, X_{n-L_j} = 1) \\
&= \sum_{k_0=0}^{\infty} \sum_{k_1=1}^{\infty} \cdots \sum_{k_j=1}^{\infty} \sum_{x_0=0}^{1} \cdots \sum_{x_{j-1}=0}^{1} (1-\tau_{x_0})^{k_0} P(X_{n-l_0} = x_0) \tau_{x_1} (1-\tau_{x_1})^{k_1-1} \\
&\qquad P(X_{n-l_1} = x_1) \ldots \tau_{x_j} (1-\tau_{x_j})^{k_j-1} P(X_{n-l_j} = x_1) P_u \\
&= \sum_{k_0=0}^{\infty} \sum_{x_0=0}^{1} \cdots \sum_{x_{j-1}=0}^{1} (1-\tau_{x_0})^{k_0} P(X_{n-l_0} = x_0) P(X_{n-l_1} = x_1) \ldots \\
&\qquad \ldots P(X_{n-l_j} = 1) P_u \\
&= \sum_{k_0=0}^{\infty} \sum_{x_0=0}^{1} (1-\tau_{x_0})^{k_0} P(X_{n-l_0} = x_0) p P_u \\
&= \left(\frac{1-p}{\tau_0} + \frac{p}{\tau_1} \right) p P_u \\
&= p
\end{aligned}$$

Hence $\langle \hat{\mu}_X \rangle = a_0 \mu_Y + p \sum_{i=1}^{N-1} a_i$. The estimator is therefore an unbiased estimator of G_1 $(= p)$ only if $a_0 \mu_Y + p \sum_{i=1}^{N-1} a_i = p$. In the absence of knowledge about μ_Y the only straightforward way to arrange this would be to take $a_0 = 0$, and $\sum_{i=1}^{N-1} a_i = 1$.

We now look at the variance of $\langle\hat{\mu}_X\rangle$. Find $\left\langle\hat{\mu}^2_{X_n}\right\rangle$,

$$\begin{aligned}\left\langle\hat{\mu}^2_{X_n}\right\rangle &= \left\langle\left(a_0X_{n-L_0}+a_1X_{n-L_1}+\cdots+a_{N-1}X_{n-L_{N-1}}\right)^2\right\rangle \\ &= \sum_{i=0}^{N-1}\sum_{j=0}^{N-1}a_ia_j\left\langle X_{n-L_i}X_{n-L_j}\right\rangle\end{aligned}$$

The double sum is split into four parts: separate the terms containing X_{n-L_0} from those that do not, take out the $X^2_{n-L_j}$ terms from both groups.

$$\begin{aligned}\left\langle\hat{\mu}^2_{X_n}\right\rangle &= a_0^2\left\langle X^2_{n-L_0}\right\rangle+2\sum_{j=1}^{N-1}a_0a_j\left\langle X_{n-L_0}X_{n-L_j}\right\rangle+\sum_{j=1}^{N-1}a_j^2\left\langle X_{n-L_j}\right\rangle+ \\ &\qquad +2\sum_{i=1}^{N-2}\sum_{j=i+1}^{N-1}a_ia_j\left\langle X_{n-L_i}X_{n-L_j}\right\rangle \\ &= a_0^2\mu_Y+2\mu_Ypa_0\sum_{j=1}^{N-1}a_j+p\sum_{j=1}^{N-1}a_j^2+2p^2\sum_{i=1}^{N-2}\sum_{j=i+1}^{N-1}a_ia_j\end{aligned}$$

since $X^2_{n-L_i}=X_{n-L_i}$. Thus the variance of $\langle\hat{\mu}_X\rangle$ becomes,

$$\begin{aligned}\left\langle(\hat{\mu}_{X_n}-\mu_X)^2\right\rangle &= a_0^2\mu_Y+2\mu_Ypa_0\sum_{j=1}^{N-1}a_j+p\sum_{j=1}^{N-1}a_j^2+2p^2\sum_{i=1}^{N-2}\sum_{j=i+1}^{N-1}a_ia_j \\ &\qquad -a_0^2\mu_Y^2-2\mu_Ypa_0\sum_{j=1}^{N-1}a_j-p^2\sum_{i=1}^{N-1}\sum_{j=1}^{N-1}a_ia_j \\ &= a_0^2\mu_Y(1-\mu_Y)+p\sum_{j=1}^{N-1}a_j^2+p^2\left(2\sum_{i=1}^{N-2}\sum_{j=i+1}^{N-1}a_ia_j-\sum_{i=1}^{N-1}\sum_{j=1}^{N-1}a_ia_j\right)\end{aligned} \tag{3.25}$$

When specializing equation (3.25) for the vector memory all $a_j = 1/N$.

$$\begin{aligned}
\left\langle (\hat{\mu}_{X_n} - \mu_X)^2 \right\rangle_v &= \frac{\mu_Y(1-\mu_Y)}{N^2} + \frac{p}{N^2}\sum_{j=1}^{N-1} 1 + \frac{p^2}{N^2}\left(2\sum_{i=1}^{N-2}\sum_{j=i+1}^{N-1} 1 - \sum_{i=1}^{N-1}\sum_{j=1}^{N-1} 1\right) \\
&= \frac{\mu_Y(1-\mu_Y)}{N^2} + \frac{p(N-1)}{N^2} + \frac{p^2}{N^2}\left((N-1)(N-2) - (N-1)^2\right) \\
&= \frac{1}{N^2}\left(\mu_Y(1-\mu_Y) + p(1-p)(N-1)\right) \\
&= \frac{1}{N^2}\left(\frac{\tau_0\tau_1 p(1-p)}{\tau_0 p + \tau_1(1-p)} + p(1-p)(N-1)\right) \\
&= \frac{N-1}{N^2}p(1-p) + \frac{\tau_0\tau_1 p(1-p)}{N^2 A} \\
&= \frac{N-1}{N^2}p(1-p) + \frac{A\sigma_Y^2}{N^2} \qquad (3.26)
\end{aligned}$$

Note that the bias for this estimator is

$$\langle \hat{\mu}_X \rangle - \mu_X = \frac{1}{N}\mu_Y + \frac{N-1}{N}P - P = \frac{\mu_Y - p}{N}$$

If we use the unbiased estimator $a_0 = 0$, $a_i = 1/N - 1$, the variance turns out to be $\frac{p(1-p)}{N-1}$.

For the scalar memory $a_j = (1-\omega)\omega^j$, $N \to \infty$.

$$\begin{aligned}
\left\langle (\hat{\mu}_{X_n} - \mu_X)^2 \right\rangle_s &= (1-\omega)^2\mu_Y(1-\mu_Y) + p(1-\omega)^2\sum_{j=1}^{\infty}\omega^{2j} + \\
&\qquad + p^2\left(2(1-\omega)^2\sum_{i=1}^{\infty}\sum_{j=i+1}^{\infty}\omega^i\omega^j - (1-\omega)^2\sum_{i=1}^{\infty}\sum_{j=1}^{\infty}\omega^i\omega^j\right) \\
&= (1-\omega)^2\mu_Y(1-\mu_Y) + p\omega^2\frac{1-\omega}{1+\omega} + p^2\left(\frac{2\omega^3}{1+\omega} - \omega^2\right) \\
&= (1-\omega)^2\mu_Y(1-\mu_Y) + p(1-p)\omega^2\frac{1-\omega}{1+\omega} \\
&= (1-\omega)^2\frac{\tau_0\tau_1 p(1-p)}{A} + p(1-p)\omega^2\frac{1-\omega}{1+\omega} \\
&= p(1-p)\omega^2\frac{1-\omega}{1+\omega} + (1-\omega)^2 A\sigma_Y^2 \qquad (3.27)
\end{aligned}$$

To find the unbiased scalar estimator of G_1 set $a_0 = 0$ and $a_i = (1-\omega)\omega^{i-1}$, this gives variance $p(1-p)\frac{1-\omega}{1+\omega}$

In the special case where $\tau_0 = \tau_1$, $a_0 \neq 0$, then $\mu_Y = p$ and,

$$\begin{aligned}
\left\langle (\hat{\mu}_{X_n} - \mu_X)^2 \right\rangle_v &= \frac{p(1-p)}{N} \\
\left\langle (\hat{\mu}_{X_n} - \mu_X)^2 \right\rangle_s &= p(1-p)\frac{1-\omega}{1+\omega}
\end{aligned}$$

Note that since $\{X_n\}$ is an iid stochastic process with expectation, $\mu_X = p$, $\sigma_x^2 = p(1-p)$, and $c(m) = 0 \; \forall \; m$, equations (3.14) and (3.18) show that if we could apply the vector and scalar memory estimators to the $\{X_n\}$ directly, we should get

$$\begin{aligned} \left\langle (\hat{\mu}_{X_n} - p)^2 \right\rangle_v &= \frac{p(1-p)}{N} \\ \left\langle (\hat{\mu}_{X_n} - p)^2 \right\rangle_s &= p(1-p)\frac{1-\omega}{1+\omega} \end{aligned}$$

Hence the estimator (3.20) seems to achieve as good results as looking at $\{X_n\}$ directly, in the equal τ case. The variance an the unbias scalar estimator is identical to $\langle(\hat{\mu}_{X_n} - p)^2\rangle_s$, and the unbias vector estimator is only slightly larger than $\langle(\hat{\mu}_{X_n} - p)^2\rangle_v$, but both are independent of τ_0, τ_1.

In this subsection we have investigated the variance of two G_1 estimation methods (the HG relationship, and G_1 direct), both using vector and scalar memories. When comparing the memory types of either method one can set N and ω to give equal variance, (example in section 4.1). To compare methods we shall look at the variance of the unbias G_1 estimator against the HG relationship.

The variance of the HG vector estimator is given by equation (3.16) which can be written as,

$$\begin{aligned} (\text{var } \mu_X)_v &= \frac{p(1-p)}{N}\frac{1}{\tau_0\tau_1}\left(1 + \frac{2}{N}\sum_{m=1}^{N-1}(N-m)(1-A)^m\right) \\ &> p(1-p)/(N) \end{aligned} \tag{3.28}$$

since $1/(\tau_0\tau_1) > 1$ and the summation is positive ($A \in [0,1]$). The unbias vector estimator has variance $p(1-p)/(N-1) \approx p(1-p)/N$, so we can say that the vector HG estimator has a greater variance. The HG scalar estimator is also greater than its unbias equivalent as $1 + \omega(1-A) > 1 - \omega(1-A)$, (see equation (3.19)).

Chapter 4

Accuracy Boolean Networks

4.1 Accuracy trees

This section looks at the performance of the accuracy heuristic on trees, i.e. networks in which no node belongs to the set of its ancestors. Thus there are no loops. For the tree each node (bar the root) has out-degree 1. As these networks have no feedback it is straight forward to calculate values of $G_{i,1}$ for each node (i is node index). For example suppose a node has two predecessors, where predecessor 1 outputs 1 with probability 0.5, predecessor 2 outputs 1 with probability 0.3. The function on the node is XOR (Boolean addition), so the probability of it outputting a 1 if it updates is 0.5. Say the node updates probabilistically, then we can use the HG relationship (equation (3.8) if the $\tau_{i,b}$'s are arbitrary) to calculate the probability of the node being in state 1, which happens also to be 0.5.

The leaves of a tree are the nodes with no predecessors (nodes $3, 8, 9, \ldots, 16$, in figure 4.1), call this set A_1. In our simulations the leaves will be given the distribution $G_{i,1}$ (i.e. they will update on every iteration), all other leaves update using state dependent update probabilities, $\tau_{i,0}$, $\tau_{i,1}$, determined by the accuracy heuristic. Given the distributions $G_{i,1}$ are independent, we can find the distributions on the nodes whose predecessors are all in A_1, (those nodes together with those in A_1, form the set A_2). This procedure can be repeated until all the nodes in the network are dealt with.

In a tree the distributions of the set A_2 are all independent. It is possible to calculate the distributions of nodes in directed acyclic graphs (DAGs). A node in a DAG can input to more than one other (but still, no node belongs to the set of its ancestors). For a DAG the distributions of set A_2 are not necessarily independent, we will need joint distributions to find the distributions for all nodes

whose predecessors are in A_1, etc.

Studying nodes 0, 1, 2 and 6 of the tree in figure 4.1 illustrates most of the key features experienced when simulating accuracy trees. Firstly we will look at the theoretical and numerical standard deviation of $\hat{G}^n_{i,1}$ (estimate of $G_{i,1}$ at n), for all four estimation methods (note, if $\hat{G}^n_{i,1}$ converges to its theoretical value then so will simulated re-evaluation probabilities $\tau^n_{i,b}$). We will then go on to talk about the accuracy, considering further examples. An external observer will be monitoring and recording the accuracy of relevant nodes.

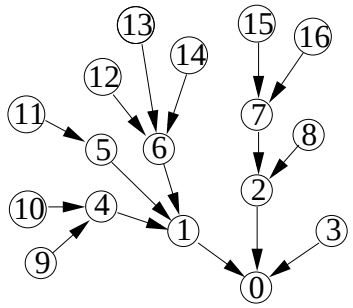

Figure 4.1: Diagram of a random Boolean tree.

To construct a tree we set a maximum in-degree and minimum number of nodes, then grow the tree from the root (node 0). Once the tree has the minimum number of nodes, leaves are assigned a $G_{i,1}$, chosen randomly. Finally all nodes with predecessors are assigned a Boolean function at random from the $2^{2^{k_i}}$ possible functions, k_i = in-degree of node i. The function takes the form of a truth table: for the tree in figure 4.1 nodes 0, 1, 2 and 6 output a 1 when the input is as shown below,

0	1	2	6
000	001		000
001	100		010
010	111		100
100			110
101			
110			

As discussed in subsection 3.2.3, there is an unbiased method for estimating $G_{i,1}$ directly (set $a_0 = 0$), giving the variances,

$$\begin{aligned} \left\langle (\hat{\mu}_{X_n} - \mu_X)^2 \right\rangle_v &= p(1-p)\frac{1}{N-1} \\ \left\langle (\hat{\mu}_{X_n} - \mu_X)^2 \right\rangle_s &= p(1-p)\frac{1-\omega}{1+\omega} \end{aligned}$$

To compare the vector and scalar estimators it would be best to set the two variances equal. We use $N = 1000$, $\omega = 0.998$, to satisfy the equality. The bias with this value of N and ω is negligible, thus in simulations we need not set $a_0 = 0$ saving computation time. We also use these values of N, ω for $G_{i,1}$ estimation when using the HG relationship. Table 4.1 confirms this is a suitable choice for comparing vector and scalar HG estimators. Let the standard deviations of the four estimators be denoted as,

$$\begin{aligned}
\sqrt{\left\langle (\hat{\mu}_{X_n} - \mu_X)^2 \right\rangle_v} &= \sigma(G, v) \\
\sqrt{\left\langle (\hat{\mu}_{X_n} - \mu_X)^2 \right\rangle_s} &= \sigma(G, s) \\
\sqrt{(\text{var } \mu_X)_v} &= \sigma(H, v) \\
\sqrt{(\text{var } \mu_X)_s} &= \sigma(H, s)
\end{aligned}$$

The theoretical values of $G_{i,1}$ for the nodes under observation are ($\epsilon = 0.1$),

Node	0	1	2	6
$G_{i,1}$	1.0	0.347	0.0	0.807

We have chosen to make $\epsilon = 0.1$ for all non-leaf nodes, however as the heuristic uses only local data there is no reason against each node having its own inaccuracy, ϵ_i. It is clear from the truth table that $G_{2,1} = 0$. Node 0 also freezes, but in state 1 because the probability of inputs 011, 111, is zero as node 2 is frozen in state 0.

The nodes of interest when considering standard deviation are 1 and 6. Table 4.1 shows the standard deviations of $G_{1,1}$ and $G_{6,1}$, calculated using equations (3.16),(3.19),(3.26) and (3.27). The numbers in table 4.1 suggest that estimating $G_{i,1}$ directly is subject to less error than using the HG relationship. In table 4.2 we have the numerical values of the standard deviation of each estimator for both nodes. Simulations ran for 50000 iterations, $\tau_{i,0}^0 = \tau_{i,1}^0 = 1$ for the first 1000 iterations so a reasonable estimation of $G_{i,1}$ could be obtained. Calculations in table 4.2 use $\hat{G}_{i,1}^{5001}$ to $\hat{G}_{i,1}^{50000}$, allowing a transient period for the network to settle. As you can see the simulated standard deviation is similar for all estimation methods, with the scalar being slightly better than the vector estimator, contradicting the theory.

When looking at the accuracy of nodes, the two frozen nodes were accurate with probability 1. This is not always true for frozen nodes. When $G_{i,b} > 1 - \epsilon$ the

node	$G_{i,1}$	standard deviation
1	0.347	$\sigma(G,v) = \sigma(G,s) = 0.015$
		$\sigma(H,v) = \sigma(H,s) = 0.025$
6	0.807	$\sigma(G,v) = \sigma(G,s) = 0.012$
		$\sigma(H,v) = \sigma(H,s) = 0.024$

Table 4.1: Theoretical values of the standard deviation of nodes 1 and 6 of the tree in figure 4.1, calculated using variance equations, (3.16),(3.19),(3.26) and (3.27).

node	$\left\langle \hat{G}^n_{i,1} \right\rangle$	standard deviation
1	0.3486	$\sigma(G,s) = 0.0131$
	0.3547	$\sigma(H,s) = 0.0148$
	0.3485	$\sigma(G,v) = 0.0167$
	0.3485	$\sigma(H,v) = 0.0172$
6	0.8076	$\sigma(G,s) = 0.0110$
	0.8079	$\sigma(H,s) = 0.0126$
	0.8096	$\sigma(H,v) = 0.0129$
	0.8095	$\sigma(G,v) = 0.0153$

Table 4.2: Numerical values of the standard deviation of nodes 1 and 6 of the tree in figure 4.1.

node is frozen in state b, but the nodes active predecessors continue to change. This means that node i will be inaccurate with probability $G_{i,\bar{b}}$, i.e. accurate with probability $G_{i,b} \in [1-\epsilon, 1]$.

The accuracy of nodes 1 and 6 are shown in figure 4.2. The vector estimator on node 6 takes longer to reach $1-\epsilon$ than its scalar comparison. For node 1 the estimators make little difference to the accuracy, however the required accuracy is never reached, it converges around 0.913.

Further simulations show that nodes whose inputs are correlated (nodes with probabilistically updating predecessors) rather than iid (nodes with leaf predecessors), tend to converge on an accuracy higher than that required. A little later (in this subsection) we calculate the correlation in $\{X^n\}$ of an arbitrary node with correlated input strings, and find it to be positive. This accounts for the increase in accuracy as a positive correlation means that if a node is in state b at n it has a greater probability of it being in state b at $n+1$ (greater than choosing the next state at random). It is not the case that correlation (accuracy) increases with A_i. This point is illustrated in the simulation shown in table 4.3, figure 4.3: the accuracy of the root is closer to $1-\epsilon$ than its predecessors. We will see

that the correlation of $\{X^n\}$ depends on the number of correlated input strings.

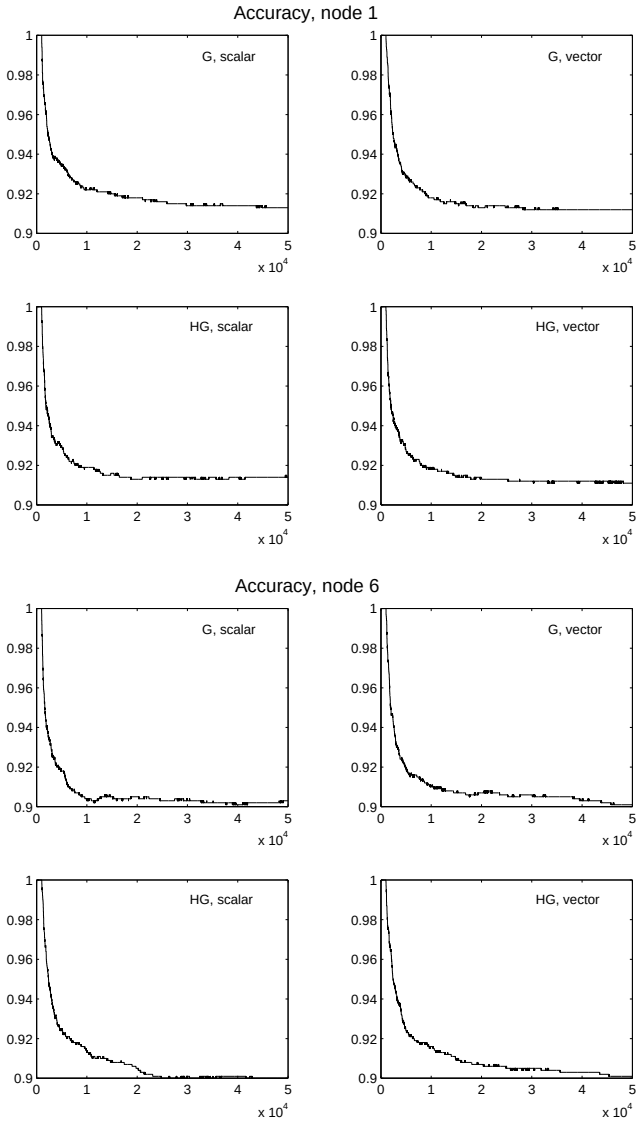

Figure 4.2: The accuracy of nodes 1 and 6 (figure 4.1) for all four estimators.

	0	1	2	3
accuracy	0.91	0.918	0.915	0.913
$G_{i,1}$	0.557	0.779	0.391	0.326
	001	010	000	000
	010	011	010	100
	011	110	110	101
	100	111		110
	111			

Table 4.3: Accuracy, $G_{i,1}$ and truth table of nodes 0 to 3 in an accuracy tree, (figure 4.3).

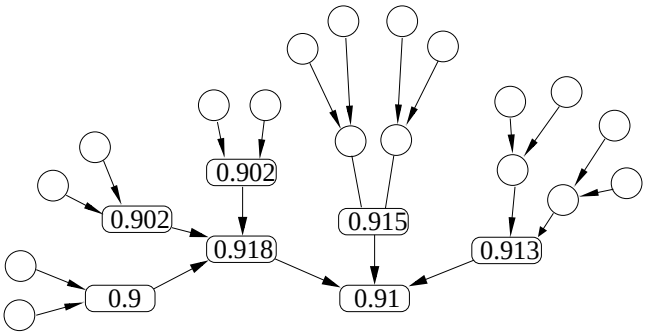

Figure 4.3: Accuracy of nodes 0 to 6 in an accuracy tree. Nodes 0 to 3 have correlated input strings (table 4.3).

In section 4.2 we will consider Boolean necklaces (loops with in-degree 1). In these networks the highly structured input strings (the structure is due to state dependent probabilistic updating and feedback effects of the loop topology) force nodes to have an accuracy close to 1 (subsection 4.2.4). For each node in such a network the input sequence $\{X_i^n\}$ is either identical with, or just the inversion of, the output of the previous node, so the amount of correlation in the output of node $i-1$ is the same as the input to node i. Nodes in trees have more than one predecessor, which serves to dilute the effects of input correlation on $\{X_i^n\}$. To give an example of this we set up a numerical experiment in which one node (node 0) has k predecessors (see figure 4.4).

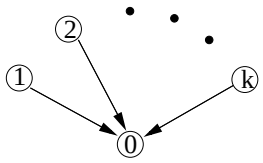

Figure 4.4: Network for the accuracy/autocorrelation experiment.

The predecessors are assigned $G_{i,1} \in (\epsilon, 1-\epsilon)$ (to prevent freezing of inputs) at random, they then calculate $\tau_{i,b}$ and update probabilistically. That is to say that when they update their state is 1 with probability $G_{i,1}$. As before we have $\epsilon = 0.1$. Node 0 has the XOR Boolean function and generates the $\{X_0^n\}$ time series (by updating on every iteration), which is stored for analysis. We calculated its autocorrelation at different lags. If the random variables in a time series are independent then it follows that their autocorrelation is zero, though the converse is not necessarily true. However if we calculate non zero autocorrelations, at least we can say that the random variables are not independent (as we have assumed

when deriving the accuracy heuristic). The autocorrelation of $\{X_0^n\}$ at lag m is estimated using,

$$r_0^m = \frac{\sum_{n=0}^{N-m-1}(X_0^n - \langle X_0^n \rangle)(X_0^{n+m} - \langle X_0^n \rangle)}{\sum_{n=0}^{N-1}(X_0^n - \langle X_0^n \rangle)^2} \tag{4.1}$$

N is the number of bits recorded for analysis, 50000 for our experiment. The results are shown in figure 4.5. We can see that as k increases the correlation at lag 1 decreases. The correlation at higher lags is insignificant for all k.

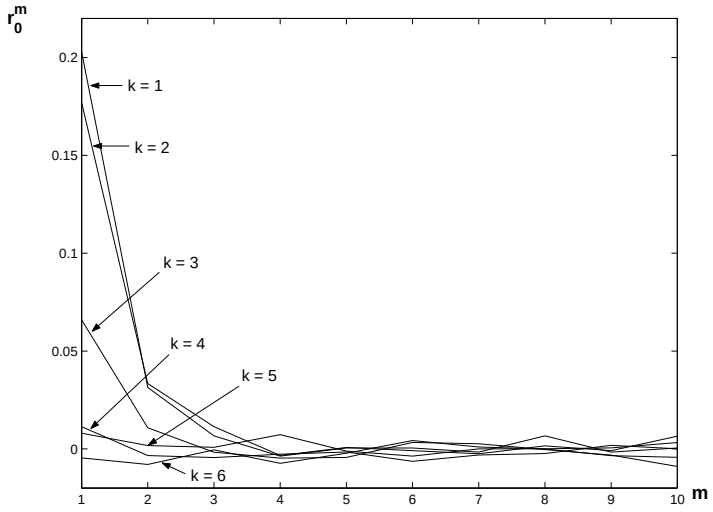

Figure 4.5: Autocorrelation of $\{X_0^n\}$ for a node with k correlated input strings.

To address the question of how correlated input effects accuracy, we performed a similar experiment, where node 0 (figure 4.4) runs the accuracy heuristic, estimating $G_{0,1}$ directly using the scalar memory, $\omega = 0.998$. Its Boolean function is still XOR. The experiment is run for 50000 iterations, $\tau_{0,b} = 1$ for the first 1000 iterations, giving a reasonable first estimate of $G_{i,1}$. The experiment was run 20 times for each k, the accuracy of node 0 was recorded and averaged to give the table below, $\epsilon = 0.1$.

in-degree	2	3	4	5	6	7	8	9	10	11
accuracy	0.923	0.910	0.906	0.905	0.905	0.904	0.903	0.902	0.902	0.902

We see that as the correlation at lag 1 decreases the accuracy of node 0 approaches its target.

4.2 Networks with in-degree one

In section 4.1 we looked at the performance of the accuracy algorithm on Boolean trees. When the state of the leaves were iid the tree's re-evaluation probabilities

could be found theoretically (given leaf distributions). However, when it comes to topologies with feedback, such as random networks, theoretical calculation of re-evaluation probabilities becomes more complicated. To help us understand the effects of feedback in an accuracy network we will look into random Boolean networks with in-degree one.

The 'activity' in RBNs with in-degree one is governed by information loops. Information loops consist only of nodes with Boolean functions f_i^1, f_i^2 (negation and identity, table 1.4) in a necklace topology. Throughout this document we will refer to loops of active nodes as information loops. Apart from loops, the only other topological structures within such RBNs are trees, and every tree is rooted in a loop. If a node has a constant Boolean function, the state if its descendants will become fixed in a finite time. For a constant node in a necklace, the whole necklace and attached trees will freeze as they are all descendants. Figure 4.6 shows two necklaces with branching trees.

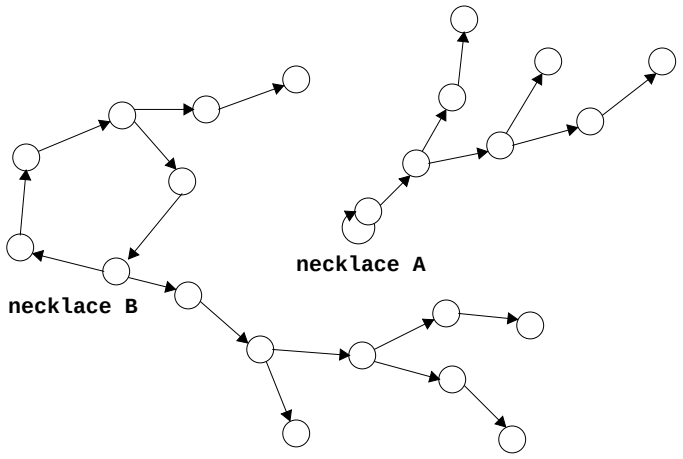

Figure 4.6: Example of a network with in-degree 1, showing necklaces with sprouting trees.

For necklace A, length 1, to be an information loop the node must host the negation function, otherwise the state of the node is frozen. A frozen state would propagate up the tree, rendering all nodes within it inactive. For necklace B to be an information loop all the nodes must be negation or identity, but this condition is not enough to ensure activity. For example, if every node in necklace B were identity then the fixed points $000\ldots0$, $111\ldots1$, would render it inactive. We will see later that loops with odd numbers of inversions can not have fixed points. Even numbers of inversions can lead to fixed points as in the following example. Label the nodes in necklace B from 0 to 4 and assign them functions,

$f_0^1, f_1^2, f_2^1, f_3^2, f_4^2$, giving the dynamical system,

$$\begin{array}{lcll} x_0^{n+1} & = & f_0^1(x_4^n) & = & \bar{x_4^n} \\ x_1^{n+1} & = & f_1^2(x_0^n) & = & x_0^n \\ x_2^{n+1} & = & f_2^1(x_1^n) & = & \bar{x_1^n} \\ x_3^{n+1} & = & f_3^2(x_2^n) & = & x_2^n \\ x_4^{n+1} & = & f_4^2(x_3^n) & = & x_3^n \end{array} \tag{4.2}$$

System (4.2) has a fixed points at $x_0x_1x_2x_3x_4 = 11000$, 00111, all other states belong to period 5 cycles, see figure 4.7. Thus, if necklace B, described by equations (4.2), is not in a fixed point, it is an information loop.

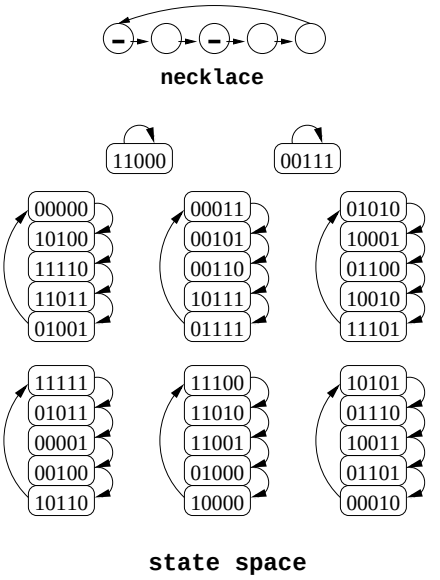

Figure 4.7: Phase portrait for system (4.2).

A necklace whose system has an odd number of inversion functions is always an information loop, it can not produce a fixed point. Consider a general necklace length L, fix node i as an inversion node, $x_i^n = \bar{x}_{i-1}^{n-1}$, (note if $i = 0$, $x_0^n = \bar{x}_{L-1}^{n-1}$). If the loop has an odd number of inversions, $2m + 1$, then there are $2m$ further inversion nodes, and $L - 2m - 1$ identity. As state x_i^n travels around the information loop it will be inverted $2m$ times over $L - 1$ iterations and so returns to its original state. Thus, $x_{i-1}^{n-1+L} = x_i^n$, giving, $x_i^{n+L} = \bar{x}_{i-1}^{n-1+L} = \bar{x}_i^n$. Following the state around again we get back to the original state. This is true for all inversion nodes and also the identity nodes. So, nodes will be in a particular state for half the time making a fixed point impossible. Figure 4.8 shows the phase portrait

for system (4.3), necklace B with inversion functions on nodes $0, 1$ and 2.

$$\begin{array}{lcll} x_0^{n+1} & = & f_0^1(x_4^n) & = & \bar{x_4^n} \\ x_1^{n+1} & = & f_1^1(x_0^n) & = & \bar{x_0^n} \\ x_2^{n+1} & = & f_2^1(x_1^n) & = & \bar{x_1^n} \\ x_3^{n+1} & = & f_3^2(x_2^n) & = & x_2^n \\ x_4^{n+1} & = & f_4^2(x_3^n) & = & x_3^n \end{array} \tag{4.3}$$

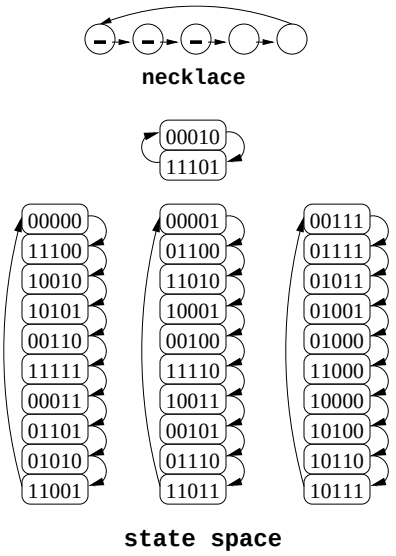

Figure 4.8: Phase portrait for system (4.3).

Because of this fundamental difference between necklaces with odd and even inversions, namely the possibility of a fixed point for even inversions, we shall consider the two cases separately from now on.

4.2.1 Number of periodic orbits, length g, in necklace state space

In analyzing the behavior of information loops a useful property to know is, how many periodic orbits does the state space of an information loop contain, and what are their lengths? In the $k = 1$ case, the state mapping of a loop is invertible (if there are only identities and inversions). It follows that every state is periodic, so the periodic orbits partition the state space.

Even inversions

Consider a necklace with L nodes, each node having a Boolean function, f_i, equal to f_i^1 (inversion) or f_i^2 (identity). Suppose there are an even number of inversion

nodes, then $x_i^{n+L} = x_i^n$ and all states are L periodic. Their prime period, g, may be less than L but must divide it. For any p such that $p|L$, we can create a state in a period p orbit by setting $x_0^0, x_1^0, \ldots, x_{p-1}^0$ arbitrarily, then use equations (4.4) to set the value of x_i for $i = p, p+1, \ldots, L-1$.

$$\begin{aligned}
x_p^0 &= f_p \circ f_{p-1} \circ \cdots \circ f_1(x_0^0) \\
x_{p+1}^0 &= f_{p+1} \circ f_p \circ \cdots \circ f_2(x_1^0) \\
&\vdots \\
x_{2p-1}^0 &= f_{2p-1} \circ f_{2p-2} \circ \cdots \circ f_p(x_{p-1}^0) \\
x_{2p}^0 &= f_{2p} \circ f_{2p-1} \circ \cdots \circ f_{p+1}(x_p^0) \\
&\vdots \\
x_{L-1}^0 &= f_{L-1} \circ f_{L-2} \circ \cdots \circ f_{L-p+1}(x_{L-p}^0)
\end{aligned} \tag{4.4}$$

Thus, for $i = p, p+1, \ldots, L-1$, $x_i^p = x_i^0$ by definition.

$$\begin{aligned}
x_i^p &= f_i \circ f_{i-1} \circ \cdots \circ f_{i-p+1}(x_{i-p}^0) \\
&= x_i^0
\end{aligned}$$

To show this holds for $i = 0, 1, \ldots, p-1$, first look at node 0.

$$x_0^p = f_0 \circ f_{L-1} \circ \cdots \circ f_{L-p+1}(x_{L-p}^0)$$

$L - p \in [p, p+1, \ldots, L-1]$, so $x_{L-p}^0 = x_{L-p}^p$

$$\begin{aligned}
x_0^p &= f_0 \circ f_{L-1} \circ \cdots \circ f_{L-p+1}(x_{L-p}^p) \\
&= f_0 \circ f_{L-1} \circ \cdots \circ f_{L-p+1}(f_{L-p} \circ f_{L-p-1} \circ \cdots \circ f_{L-2p+1}(x_{L-2p}^0))
\end{aligned}$$

Likewise $x_{L-mp}^0 = x_{L-mp}^p$ as $L - mp \in [p, p+1, \ldots, L-1]$ up until $L - mp = p$.

$$\begin{aligned}
x_0^p &= f_0 \circ f_{L-1} \circ \cdots \circ f_{L-2p+1}(x_{L-2p}^p) \\
&= f_0 \circ f_{L-1} \circ \cdots \circ f_{L-3p+1}(x_{L-3p}^p) \\
&\cdots \\
&= \underbrace{f_0 \circ f_{L-1} \circ \cdots \circ f_1}_{f^1}(x_0^0) \\
&= x_0^0
\end{aligned}$$

Similar arguments can be made for $i = 1, 2, \ldots, p-1$ and so the state $x_0 x_1 \ldots x_{L-1}$, generated using this method is p periodic. There are 2^p possible p periodic states as they are completely defined by fixing p consecutive states, in particular there are two fixed points for $p > 1$. The 2^p states can not all have prime period p as

the count will include all points of period q, such that $q|p$, most noticeably it will contain the fixed points. Let $g(p)$ be the number of states with prime period p. Then the number of states with period p is given by the formula,

$$2^p = \sum_{d|p} g(d)$$

for all p such that $p|L$. We can invert this using the Möbius inversion formula [29] [28],

$$g(p) = \sum_{d|p} \mu\left(\frac{p}{d}\right) 2^d \tag{4.5}$$

$$\text{where,} \quad \mu(a) = \begin{cases} 1 & \text{if } a = 1 \\ 0 & \text{if } \rho^2|a \quad \text{for some prime } \rho \\ (-1)^r & \text{if } a = \rho_1, \rho_2, \ldots, \rho_r \quad \text{where } \rho_i \text{ are distinct primes} \end{cases}$$

Equation (4.5) gives the number of states with prime period $g(p)$, it does not depend on f_i^1 past the fact that inversions are even in number. We see that the number and length of cycles in the state space of an even inversion necklace depends only on its length, i.e. all even inversion necklaces of length L, have the same number of period p orbits.

eg. Calculate the number and length of cycles in a necklace length 5, with an even number of inversions.
The only two divisors of 5 are, 1 and 5. The number of states with prime period 1 are,

$$\begin{aligned} g(1) &= \mu(1)\, 2 \\ &= 2 \end{aligned}$$

There are two fixed points. The number of states with prime period 5 are,

$$\begin{aligned} g(5) &= \sum_{d|5} \mu\left(\frac{5}{d}\right) 2^d \\ &= \mu(5)\, 2 + \mu(1)\, 32 \\ &= -2 + 32 \\ &= 30 \end{aligned}$$

Giving, six period 5 orbits, see (figure 4.7).

Odd inversions

As discussed earlier, there are no period one orbits in necklaces with an odd number of inversion nodes, they are all information loops. Furthermore, $x_i^{n+L} =$

$\bar{x}_i^n$, every state has period $2L$ and so we are looking for period p orbits where $p|2L$ but not L. Using prime decomposition we can write L, $2L$ and p as a multiplication of primes, ρ_i,

$$\begin{aligned} L &= 2^{\alpha_1}\rho_2^{\alpha_2}\rho_3^{\alpha_3}\dots \\ 2L &= 2^{\alpha_1+1}\rho_2^{\alpha_2}\rho_3^{\alpha_3}\dots \\ p &= 2^{\beta_1}\rho_2^{\beta_2}\rho_3^{\beta_3}\dots \end{aligned}$$

where, $\rho_2 = 3$, $\rho_3 = 5$, etc. and $\alpha_i, \beta_i = 0, 1, 2, \dots$. If $p|2L$ but not L, $\alpha_1 < \beta_1 \leq \alpha_1 + 1 \Rightarrow \beta_1 = \alpha_1 + 1$. Thus,

$$\begin{aligned} 2L &= 2^{\alpha_1+1}\rho_2^{\alpha_2}\rho_3^{\alpha_3}\dots &&= 2^{\alpha_1+1}L' \\ p &= 2^{\alpha_1+1}\rho_2^{\beta_2}\rho_3^{\beta_3}\dots &&= 2^{\alpha_1+1}m \quad \text{where} \quad m|L' \end{aligned}$$

We will show that there are $2^{p/2}$ states with period p, let $q = p/2$. To count the number of states with prime period p construct a similar argument as for the even inversion case. Note that L is an odd multiple of q,

$$\frac{L}{q} = \frac{2L}{p} = \frac{L'}{m} = \rho_2^{\gamma_2}\rho_3^{\gamma_3}\rho_4^{\gamma_4}\dots$$

$\gamma_i = \alpha_i - \beta_i$

Assign $x_0^0, x_1^0, \dots, x_{q-1}^0$ arbitrarily. Using equations (4.6) we set the values of x_i^0 in blocks, length q, starting with x_{mq-1}^0, m even, until we set x_{L-1}^0.

$$\begin{aligned} x_{2q}^0 &= f_{2q} \circ f_{2q-1} \circ \cdots \circ f_1(x_0^0) \\ x_{2q+1}^0 &= f_{2q+1} \circ f_{2q} \circ \cdots \circ f_2(x_1^0) \\ &\vdots \\ x_{3q-1}^0 &= f_{3q-1} \circ f_{3q-2} \circ \cdots \circ f_q(x_{q-1}^0) \\ x_{4q}^0 &= f_{4q} \circ f_{4q-1} \circ \cdots \circ f_{2q+1}(x_{2q}^0) \\ &\vdots \\ x_{L-1}^0 &= f_{L-1} \circ f_{L-2} \circ \cdots \circ f_{L-2q}(x_{L-1-2q}^0) \end{aligned} \tag{4.6}$$

Now the remaining q length blocks can be calculated as follows:

$$\begin{aligned} x_q^0 &= f_q \circ f_{q-1} \circ \cdots \circ f_0 \circ f_{L-1} \circ f_{L-2} \circ \cdots \circ f_{L+1-q}(x_{L-q}^0) \\ x_{q+1}^0 &= f_{q+1} \circ f_q \circ \cdots \circ f_{L+2-q}(x_{L+1-q}^0) \\ &\vdots \\ x_{2q-1}^0 &= f_{2q-1} \circ f_{2q-2} \circ \cdots \circ f_0(x_{L-1}^0) \\ x_{3q}^0 &= f_{3q} \circ f_{3q-1} \circ \cdots \circ f_{q-1}(x_q^0) \\ &\vdots \\ x_{L-1-q}^0 &= f_{L-1-q} \circ f_{L-2-q} \circ \cdots \circ f_{L-3q}(x_{L-1-3q}^0) \end{aligned}$$

Thus, by definition, $x_i^{2q} = x_i^p = x_i^0$ for $i = q, q+1, \ldots, L-1$. To show this holds for the first q nodes is an identical argument to the even case: Consider node 0,

$$\begin{aligned}
x_0^{2q} &= f_0 \circ f_{L-1} \circ \cdots \circ f_{L-2q+1}(x_{L-2q}^0) \\
&= f_0 \circ f_{L-1} \circ \cdots \circ f_{L-2q+1}(x_{L-2q}^{2q}) \\
&= f_0 \circ f_{L-1} \circ \cdots \circ f_{L-2q+1}(f_{L-2q} \circ f_{L-2q-1} \circ \cdots \circ f_{L-4q+1}(x_{L-4q}^0)) \\
&= f_0 \circ f_{L-1} \circ \cdots \circ f_{L-4q+1}(x_{L-4q}^{2q}) \\
&= f_0 \circ f_{L-1} \circ \cdots \circ f_{L-6q+1}(x_{L-6q}^{2q}) \\
&\vdots \\
&= f_0 \circ f_{L-1} \circ \cdots \circ f_{q+1}(x_q^{2q})
\end{aligned}$$

Recall L is an odd multiple of q, so,

$$\begin{aligned}
x_0^{2q} &= f_0 \circ f_{L-1} \circ \cdots \circ f_{q+1}(f_q \circ f_{q-1} \circ \cdots \circ f_0 \circ f_{L-1} \circ \cdots \circ f_{L-q+1}(x_{L-q}^0)) \\
&= f_0 \circ f_{L-1} \circ \cdots \circ f_0 \circ f_{L-1} \circ \cdots \circ f_{L-q+1}(x_{L-q}^{2q}) \\
&= f_0 \circ f_{L-1} \circ \cdots \circ f_0 \circ f_{L-1} \circ \cdots \circ f_{L-3q+1}(x_{L-3q}^{2q}) \\
&\vdots \\
&= \underbrace{f_0 \circ f_{L-1} \circ \cdots \circ f_0 \circ f_{L-1} \circ \cdots \circ f_1}_{f^1}(x_0^0) \\
&= x_0^0
\end{aligned}$$

Similarly $x_1, x_2, \ldots, x_{q-1}$ can be shown to be period p points.

As we set q values arbitrarily, there are 2^q period p points. Prime periods must be of the form $2^{\alpha_1+1}m'$ where $m'|m$. This ensures that the prime period divides p but not L. The number of states with period p is given by

$$2^{p/2} = 2^{2^{\alpha_1} m} = \sum_{m'|m} g(m')$$

Note that, $g(m')$ is the number of states with period $2^{\alpha_1+1}m'$, a slightly different definition to the even case. Using the Möbius inversion formula we get,

$$g(m) = \sum_{m'|m} \mu\left(\frac{m}{m'}\right) 2^{2^{\alpha_1} m'}$$

which gives us the number of states with prime period $2^{\alpha_1+1}m'$. We see that $g(m)$ depends on the length of the information loop and inversion nodes must be odd in number.

4.2.2 Conjugacy, mapping even inversion necklaces to a no inversion necklace, and odd inversion necklaces to a one inversion necklace

In subsection 4.2.1 we saw that, numbers of periodic orbits depend on the length of the necklace and whether the necklace has an odd or even number of inversion nodes. As necklaces with even/odd inversions, length L, have the same number of period g orbits, one may suspect that the state space of all even/odd necklaces, length L, may be mapped to one another. Here we are going to define a conjugacy, T, which maps all odd inversion loops to a one inversion loop of the same length, and all even inversion necklaces to a no inversion necklace of the same length.

As we have seen, the state of a necklace is updated using a system of equations. Define F_0, F_1, F_{evn} and F_{odd} as systems made up of L Boolean functions, f_i^j, with zero, one, an even or odd number of inversion functions, respectively. The state space of the dynamical system generated by these equations is $\mathbf{B}^L = \{0,1\}^L$ in all cases, but we will distinguish each space, $\mathbf{B}_0^L$, $\mathbf{B}_1^L$, $\mathbf{B}_{evn}^L$, $\mathbf{B}_{odd}^L$. For convenience we will represent F_* as a Boolean vector, length L, where 1 in position i signifies node i is an inversion node. This notation is particularly fitting as $1 \oplus b = \bar{b}$ and $0 \oplus b = b$ ($\oplus$ is Boolean addition).

$$\begin{array}{rcl} F_0 &=& [0,0,0,\ldots,0] \\ F_1 &=& [1,0,0,\ldots,0] \end{array} \qquad \begin{array}{l} F_0: \ \mathbf{B}_0^L \to \mathbf{B}_0^L \\ F_1: \ \mathbf{B}_1^L \to \mathbf{B}_1^L \end{array}$$

$$F_{evn} = [g_0, g_1, g_2, \ldots, g_{L-1}], \quad \bigoplus_{i=0}^{L-1} g_i = 0, \quad F_{evn}: \ \mathbf{B}_{evn}^L \to \mathbf{B}_{evn}^L$$

$$F_{odd} = [g_0, g_1, g_2, \ldots, g_{L-1}], \quad \bigoplus_{i=0}^{L-1} g_i = 1, \quad F_{odd}: \ \mathbf{B}_{odd}^L \to \mathbf{B}_{odd}^L$$

To illustrate how the F vector function systems are used, let
$X^n = [x_0^n, x_1^n, x_2^n, \ldots, x_{L-1}^n] \in \mathbf{B}_*^L$, and look at F_1, F_{odd},

$$\begin{array}{rcl} F_1(X^n) &=& X^{n+1} = [\bar{x}_{L-1}^n, x_0^n, x_1^n, \ldots, x_{L-2}^n] \\ F_{odd}(X^n) &=& X^{n+1} = [g_0 \oplus x_{L-1}^n, g_1 \oplus x_0^n, g_2 \oplus x_1^n, \ldots, g_{L-1} \oplus x_{L-2}^n] \end{array}$$

Define the transformation T,

$$T = [h_0, h_1, h_2, \ldots, h_{L-1}], \qquad \begin{array}{l} T: \mathbf{B}_{evn}^L \to \mathbf{B}_0^L \\ T: \mathbf{B}_{odd}^L \to \mathbf{B}_1^L \end{array}$$

$$h_i = \bigoplus_{j=0}^{i} g_j$$

$$T(X^n) = [h_0 \oplus x_0^n, h_1 \oplus x_1^n, h_2 \oplus x_2^n, \ldots, h_{L-1} \oplus x_{L-1}^n]$$

Note that $T^{-1} = T$ as, $T \circ T = T \oplus T = [0, 0, 0, \ldots, 0]$.

Recall, we are interested in studying the behavior of information loops in the hope of gaining some insight into the feedback effects on the accuracy heuristic. If T is a conjugacy then any dynamical characteristics we find in loops with zero/one inversion node will hold for all even/odd inversion loops. We will now show the property of conjugacy holds, namely,

$$T \circ F_0(X^n) = F_{evn} \circ T(X^n) \tag{4.7}$$
$$T \circ F_1(X^n) = F_{odd} \circ T(X^n) \tag{4.8}$$

First we will look at condition (4.7), for the even inversion node case.

$$\begin{aligned} T \circ F_0(X^n) &= T([x^n_{L-1}, x^n_0, x^n_1, \ldots, x^n_{L-2}]) \\ &= [h_0 \oplus x^n_{L-1}, h_1 \oplus x^n_0, h_2 \oplus x^n_1, \ldots, h_{L-1} \oplus x^n_{L-2}] \end{aligned}$$

$$\begin{aligned} F_{evn} \circ T(X^n) &= F_{evn}([h_0 \oplus x^n_0, h_1 \oplus x^n_1, h_2 \oplus x^n_2, \ldots, h_{L-1} \oplus x^n_{L-1}]) \\ &= [g_0 \oplus h_{L-1} \oplus x^n_{L-1}, g_1 \oplus h_0 \oplus x^n_0, g_2 \oplus h_1 \oplus x^n_1, \ldots, g_{L-1} \oplus h_{L-2} \oplus x^n_{L-2}] \end{aligned}$$

$h_{L-1} = 0$ for even inversion necklaces, $h_0 = g_0$, and $g_i \oplus h_{i-1} = h_i$. Thus,

$$\begin{aligned} F_{evn} \circ T(X^n) &= [h_0 \oplus x^n_{L-1}, h_1 \oplus x^n_0, h_2 \oplus x^n_1, \ldots, h_{L-1} \oplus x^n_{L-2}] \\ &= T \circ F_0(X^n) \end{aligned}$$

Condition (4.7) holds. To show condition (4.8) holds we will use the fact that $h_{L-1} = 1$ for even inversion loops.

$$\begin{aligned} T \circ F_1(X^n) &= T([\bar{x}^n_{L-1}, x^n_0, x^n_1, \ldots, x^n_{L-2}]) \\ &= [h_0 \oplus \bar{x}^n_{L-1}, h_1 \oplus x^n_0, h_2 \oplus x^n_1, \ldots, h_{L-1} \oplus x^n_{L-2}] \end{aligned}$$

$$\begin{aligned} F_{odd} \circ T(X^n) &= F_{odd}([h_0 \oplus x^n_0, h_1 \oplus x^n_1, h_2 \oplus x^n_2, \ldots, h_{L-1} \oplus x^n_{L-1}]) \\ &= [g_0 \oplus h_{L-1} \oplus x^n_{L-1}, g_1 \oplus h_0 \oplus x^n_0, g_2 \oplus h_1 \oplus x^n_1, \ldots, g_{L-1} \oplus h_{L-2} \oplus x^n_{L-2}] \\ &= [h_0 \oplus 1 \oplus x^n_{L-1}, h_1 \oplus x^n_0, h_2 \oplus x^n_1, \ldots, h_{L-1} \oplus x^n_{L-2}] \\ &= [h_0 \oplus \bar{x}^n_{L-1}, h_1 \oplus x^n_0, h_2 \oplus x^n_1, \ldots, h_{L-1} \oplus x^n_{L-2}] \\ &= T \circ F_1(X^n) \end{aligned}$$

T is a conjugacy [1].

[1] Proof by Nick Watson, `www.pipie.co.uk`

4.2.3 Accuracy information loops

When investigating the effect of the accuracy heuristic on information loops we are going to study the evolution of $\tau_{i,b}$, which depends on $G_{i,b}$. As we have discussed, if $\tau_{i,b} \neq 1$ there are two ways to estimate $G_{i,b}$: we either observe $H_{i,b}$ then use equation (3.8) to estimate $G_{i,b}$, or we estimate $G_{i,b}$ directly (see subsection 3.2.1). The HG relationship described by equation (3.8) is not appropriate when simulating loops as it depends on the current input being independent of the current state. The loop structure means that input must depend on previous states, so the current state will not be independent of it. Thus, when simulating accuracy information loops we must estimate $G_{i,b}$ directly.

In fact when analyzing the behaviour of an information loop we have an exact value for $G_{i,b}$. Recall, $G_{i,b}$ is the distribution of node i if it were to update on every iteration, i.e. if $\tau_{i,b} = 1$. It is clear that $G_{i,b} = H_{i-1,b}$ if node i is an identity node, or $1 - H_{i-1,b}$ if it is an inversion, (see figure 4.9).

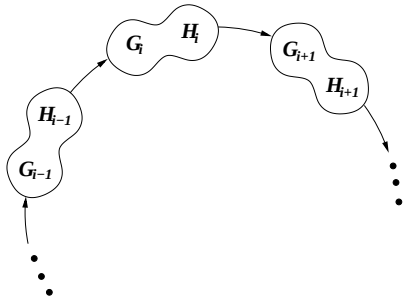

Figure 4.9: Detail of nodes in an accuracy information loop.

Even inversion accuracy loops

The job of investigating the long term behaviour of accuracy information loops, with no inversion nodes, is simplified by the realization that once the accuracy heuristic is imposed on the information loop it gets drawn into one of the two fixed points, $[000\ldots0]$ or $[111\ldots1]$. To illustrate why this is look at the example of a state in $\mathbf{B}_0^{12}$.

$$\begin{array}{cc} X^n & [0\underline{0000}101\underline{11}01] \\ F_0(X^n) & [100000101110] \end{array}$$

It is not the case that any state in accuracy state space can lead to any another, in fact movement around accuracy state space is guided by the deterministic state space. This is clear from the above example. In particular, we see that all bits in

a block, bar the first, are fixed during an iteration of F_0, (underlined in example). Thus, when probabilistic updating is enforced, blocks can shrink if the first node of the block updates, but they can also grow if the node to the right of the block updates.

Re-evaluation probabilities will influence the growth of blocks. If the loop in our example ran deterministically (to establish $G^0_{i,b}$, and hence $\tau^1_{i,b}$), we would have $G_{i,0} \approx 7/12$, $G_{i,1} \approx 5/12$. Equation (3.7) would then lead to $\tau^1_{i,0} < \tau^1_{i,1}$, so if we started to update probabilistically a node in state zero will update less frequently, promoting the growth of zero blocks. As the zero blocks grow, $\tau^n_{i,0}$ will decrease further, guiding the loop to the zero fixed point. If the re-evaluation probabilities were equal, block size is the major contributory factor when determining which fixed point the loop converges to. The important point is that blocks can shrink to zero size, and so disappear, but new blocks can never appear. The number of blocks either remains the same or decreases. Eventually it decreases to one.

Odd inversion accuracy loops

Investigating the behaviour of probabilistic information loops with one inversion node, (say node 0) is also simplified by observing that the state space contains a cycle (which is one of the periodic orbits of the deterministic system) that absorbs all other initial conditions. Once the cycle has been reached the trajectory visits states on the cycle in a particular order, although the network may remain in any one of these states for a number of iterations.

If a state X^n and its (deterministic) successor $F_1(X^n)$ are separated by Hamming distance H, then the number of possible successors of X^n under probabilistic updating is 2^H - and one of these is X^n itself. A deterministic information loop of length L has a periodic orbit of the form shown in figure 4.10. Note the distance between each state and the next is 1, so each state has 2 possible successors under probabilistic updating, one of which is itself. Hence any trajectory that reaches this orbit remains on it for all time.

It is also clear that any state can reach the orbit by making transitions that have a finite non-zero probability. For example, if node 0 fails to update for $L-1$ iterations, but all the other nodes do update, the network will be in state $000\ldots0$ or $111\ldots1$. So the invariant density for the states will all be concentrated on the periodic orbit.

$\vdots$	$\vdots$
0	$000\ldots00$
1	$100\ldots00$
2	$110\ldots00$
$\vdots$	$\vdots$
$L-1$	$111\ldots10$
L	$111\ldots11$
$L+1$	$011\ldots11$
$L+2$	$001\ldots11$
$\vdots$	$\vdots$
$2L-1$	$000\ldots01$
0	$000\ldots00$
$\vdots$	$\vdots$

Figure 4.10: The right column shows states in the limit cycle of one inversion accuracy information loop, length L. The left column gives the label of each state.

The conjugacy between a loop with an odd number of inversions and the loop with one inversion (see subsection 4.2.2) means there is a corresponding orbit of length $2L$ in all such networks. This is because the conjugacy map preserves distance as it simply flips certain bits within the states. T will flip the same bits in X^n and X^{n+1}, so if the bits are the same in X^n, X^{n+1}, they will be the same in $T(X^n)$ and $T(X^{n+1})$, leaving the distance unaltered.

As $x_i^{n+L} = \bar{x}_i^n$, $x_i^{n+2L} = x_i^n$, in odd inversion loops within the deterministic regime, the value of $G_{i,b}$ is 0.5. Thus, the first re-evaluation probabilities with a value less than 1 will be $\tau_{i,0} = \tau_{i,1} = 1 - 2\epsilon$. We will now show this to be a fixed point, in the sense that if we assign these values to $\tau_{i,b}$, compute the corresponding $G_{i,b}$'s, and use these to find new $\tau_{i,b}$ values, these values remain at $1-2\epsilon$. We will also investigate the stability of the fixed point.

Denote $\mathcal{F}_1^n : \mathbf{B}_1^L \to \mathbf{B}_1^L$ to be the accuracy map on the information loop with one inversion node. To investigate the long term behavior of $\mathcal{F}_1^n(X^n)$ we are going to exploit the fact that movement around accuracy state space is a Markov process. To do this efficiently requires an alteration to our notation. Let,

$$\tau_{(k)} \quad = \quad \begin{cases} \tau_{k,0} & \text{for} \quad k = 0, 1, 2, \ldots, L-1 \\ \tau_{k-L,1} & \text{for} \quad k = L, L+1, L+2, \ldots, 2L-1 \end{cases}$$

The probability that state k (labelling as in figure 4.10) moves to state $k+1$

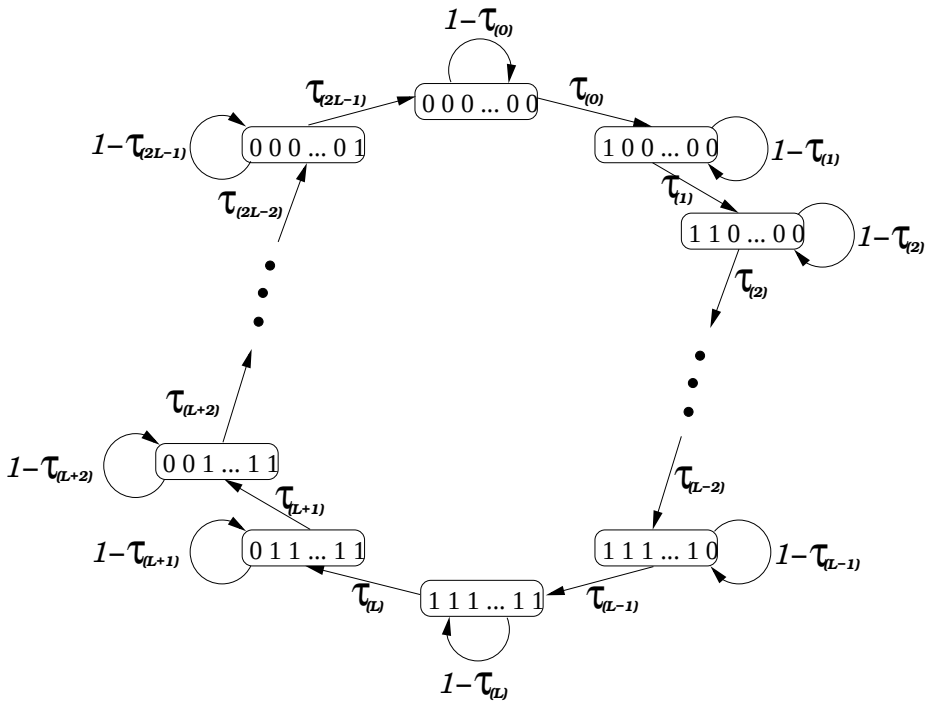

Figure 4.11: Transition Diagram for the limit cycle of an accuracy information loop with one inversion node, node 0.

is the probability that node k (or node $k-L$ if $L \leq k \leq 2L-1$) updates, which is $\tau_{i,0}$ for $0 \leq k \leq L-1$ and $\tau_{k-L,1}$ for $L \leq k \leq 2L-1$; hence always $\tau_{(k)}$.

We can use the transition matrix of the limit cycle (figure 4.11) to certify that $\tau_{(k)} = 1-2\epsilon$ is a fixed point. $\mu_{(k)}$ = probability of being in a particular state k, where the states are labelled as in figure 4.10. And so the Markov eigenvector equation for the $\mu_{(k)}$'s is,

$$\begin{bmatrix} \mu_{(0)} \\ \mu_{(1)} \\ \mu_{(2)} \\ \mu_{(3)} \\ \vdots \\ \mu_{(2(L-1))} \\ \mu_{(2L-1)} \end{bmatrix} = \begin{bmatrix} 1-\tau_{(0)} & 0 & \dots & 0 & \tau_{(2L-1)} \\ \tau_{(0)} & 1-\tau_{(1)} & \dots & 0 & 0 \\ 0 & \tau_{(1)} & \dots & 0 & 0 \\ 0 & 0 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & 1-\tau_{(2(L-1))} & 0 \\ 0 & 0 & \dots & \tau_{(2(L-1))} & 1-\tau_{(2L-1)} \end{bmatrix} \begin{bmatrix} \mu_{(0)} \\ \mu_{(1)} \\ \mu_{(2)} \\ \mu_{(3)} \\ \vdots \\ \mu_{(2(L-1))} \\ \mu_{(2L-1)} \end{bmatrix} \tag{4.9}$$

Equations (4.9) give us $2L-1$ independent equations with $2L$ unknowns but we can also use,

$$\sum_{k=0}^{2L-1} \mu_{(k)} = 1 \tag{4.10}$$

From the first equation of (4.9), we get,

$$\begin{aligned} \mu_{(0)} &= (1-\tau_{(0)})\mu_{(0)} + \tau_{(2L-1)}\mu_{(2L-1)} \\ \tau_{(0)}\mu_{(0)} &= \tau_{(2L-1)}\mu_{(2L-1)} \end{aligned}$$

Now look at the general case,

$$\begin{aligned} \mu_{(k)} &= (1-\tau_{(k)})\mu_{(k)} + \tau_{(k-1)}\mu_{(k-1)} \\ \tau_{(k)}\mu_{(k)} &= \tau_{(k-1)}\mu_{(k-1)} \end{aligned}$$

We can find $\tau_{(k)}$ in terms of $\tau_{(0)}$,

$$\tau_{(k)} = \frac{\mu_{(k-1)}}{\mu_{(k)}}\tau_{(k-1)} = \frac{\mu_{(k-1)}}{\mu_{(k)}}\frac{\mu_{(k-2)}}{\mu_{(k-1)}}\tau_{(k-2)} = \frac{\mu_{(k-2)}}{\mu_{(k)}}\frac{\mu_{(k-3)}}{\mu_{(k-2)}}\tau_{(k-3)} = \cdots = \frac{\mu_{(0)}}{\mu_{(k)}}\tau_{(0)} \tag{4.11}$$

Let $\rho_{(k)} = \frac{1}{\tau_{(k)}}$, and sub equation (4.11) into equation (4.10) to get,

$$1 = \tau_{(0)}\mu_{(0)} \sum_{k=0}^{2L-1} \rho_{(k)} = \tau_{(0)}\mu_{(0)}c$$

where, $c = \sum_{k=0}^{2L-1} \rho_{(k)}$.

$$\mu_{(0)} = \frac{\rho_{(0)}}{c} \quad \Rightarrow \quad \mu_{(k)} = \frac{\rho_{(k)}}{c}$$

It is straight forward to check that $\tau_{(k)} = 1-2\epsilon$ solves equations (4.9) and (4.10), thus $\tau_{(k)} = 1-2\epsilon$ is indeed a fixed point of the accuracy information loop. We will now investigate its stability.

Fix $\tau_{(k)}$ and measure $H_{(k)}$ (the probability of a node i being in state b, $H_{i,b}$), where we use the labelling,

$$H_{(k)} = \begin{cases} H_{k,0} & \text{for} \quad k = 0, 1, 2, \ldots, L-1 \\ H_{k-L,1} & \text{for} \quad k = L, L+1, L+2, \ldots, 2L-1 \end{cases}$$

We then know $G_{(k)}$ because, $G_{(k)} = H_{(k-1)}$. For convenience we extend the range of the label k by saying, $\mu_{(k)} = \mu_{(k\pm 2L)}$, $\tau_{(k)} = \tau_{(k\pm 2L)}$, $H_{(k)} = H_{(k\pm 2L)}$ and $G_{(k)} = G_{(k\pm 2L)}$ (see figure 4.9). Using the values of $G_{(k)}$ we can update the re-evaluation probabilities to get,

$$\begin{aligned} \hat{\tau}_{(k)} &= 1 - \frac{\epsilon}{1-G_{(k)}} \\ &= 1 - \frac{\epsilon}{1-H_{(k-1)}} \\ \hat{\rho}_{(k)} &= \frac{1-H_{(k-1)}}{1-H_{(k-1)}-\epsilon} \end{aligned} \tag{4.12}$$

We have an expression for $\mu_{(k)}$ in terms of $\rho_{(k)}$, and so we can write an expression for $H_{(k)}$ in terms of $\rho_{(k)}$ (refer to figure 4.10),

$$\begin{aligned} H_{(k)} &= \sum_{j=k-L+1}^{k} \mu_{(j)} \\ &= \frac{1}{c} \sum_{j=k-L+1}^{k} \rho_{(j)} \end{aligned}$$

And so equation (4.12) becomes,

$$\begin{aligned} \hat{\rho}_{(k)} &= \frac{1 - \frac{1}{c}\sum_{j=k-L}^{k-1} \rho_{(j)}}{1 - \frac{1}{c}\sum_{j=k-L}^{k-1} \rho_{(j)} - \epsilon} \\ &= \frac{c - \sum_{j=k-L}^{k-1} \rho_{(j)}}{c - \sum_{j=k-L}^{k-1} \rho_{(j)} - c\epsilon} \\ &= \frac{\sum_{j=k}^{k+L-1} \rho_{(j)}}{\sum_{j=k}^{k+L-1} \rho_{(j)} - c\epsilon} \end{aligned} \tag{4.13}$$

Equation (4.13) tells us how to find the new update probabilities from the old; applying the accuracy heuristic thus corresponds to iterating this equation. (Note that (4.13) uses the exact $G_{i,b}$'s corresponding to a given set of $\tau_{i,b}$'s).

We are now in a position to generate the Jacobian of equation (4.13) and investigate the stability of $\rho_{(k)} = 1/(1-2\epsilon)$.

Obviously

$$\frac{\partial}{\partial \rho_i} \left(\sum_{j=k}^{k+L-1} \rho_j \right) = \begin{cases} 1 & \text{if } i \in \{k, k+1, \ldots, k+L-1\} \\ 0 & \text{otherwise} \end{cases}$$

and

$$\frac{\partial}{\partial \rho_{(i)}} \left(\sum_{j=k}^{k+L-1} \rho_{(j)} - \epsilon \sum_{j=0}^{2L-1} \rho_{(j)} \right) = \begin{cases} 1-\epsilon & \text{if } i \in \{k, k+1, \ldots, k+L-1\} \\ -\epsilon & \text{otherwise} \end{cases}$$

so,

$$\left. \frac{\partial \hat{\rho}_{(k)}}{\partial \rho_{(i)}} \right|_{1/1-2\epsilon} = \begin{cases} \frac{-\epsilon}{L(1-2\epsilon)} & \text{if } i \in \{k, k+1, \ldots, k+L-1\} \\ \\ \frac{\epsilon}{L(1-2\epsilon)} & \text{otherwise} \end{cases}$$

Giving the Jacobian,

$$J = \begin{bmatrix} \frac{\partial\hat{\rho}_{(0)}}{\partial\rho_{(0)}} & \frac{\partial\hat{\rho}_{(0)}}{\partial\rho_{(1)}} & \cdots & \frac{\partial\hat{\rho}_{(0)}}{\partial\rho_{(2L-1)}} \\ \frac{\partial\hat{\rho}_{(1)}}{\partial\rho_{(0)}} & \frac{\partial\hat{\rho}_{(1)}}{\partial\rho_{(1)}} & \cdots & \frac{\partial\hat{\rho}_{(1)}}{\partial\rho_{(2L-1)}} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial\hat{\rho}_{(2L-1)}}{\partial\rho_{(0)}} & \frac{\partial\hat{\rho}_{(2L-1)}}{\partial\rho_{(1)}} & \cdots & \frac{\partial\hat{\rho}_{(2L-1)}}{\partial\rho_{(2L-1)}} \end{bmatrix}$$

$$= \frac{\epsilon}{L(1-2\epsilon)} \begin{bmatrix} -1 & -1 & \ldots & -1 & 1 & 1 & \ldots & 1 \\ 1 & -1 & \ldots & -1 & -1 & 1 & \ldots & 1 \\ 1 & 1 & \ldots & -1 & -1 & -1 & \ldots & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ -1 & -1 & \ldots & 1 & 1 & 1 & \ldots & -1 \end{bmatrix}$$

J is a circulant matrix and so its eigenvalues have a general equation which depends on its first row and the n'th roots of unity [5]. If $(c_0, c_1, \ldots, c_{n-1})$ is the first row of an $n \times n$ circulant matrix the eigenvalues are given by, $\lambda_j = c_0 + c_1\theta_j + c_2\theta_j^2 + \cdots + c_{n-1}\theta_j^{n-1}$, where $\theta_j = e^{2\pi ij/n}$, $j = 1, 2, 3, \ldots, n$. Here we have $c_j = \pm 1$, $\theta_j = e^{\pi ij/L}$.

$$\begin{aligned} \lambda_j &= \frac{\epsilon}{L(1-2\epsilon)} \left(\sum_{k=0}^{L-1} \theta_j^{L+k} - \sum_{k=0}^{L-1} \theta_j^k \right) \\ &= \frac{\epsilon}{L(1-2\epsilon)} \left(\frac{\theta^L(1-\theta_j^L)}{(1-\theta_j)} - \frac{(1-\theta_j^L)}{(1-\theta_j)} \right) \\ &= \frac{-\epsilon(1-\theta_j^L)^2}{L(1-2\epsilon)(1-\theta_j)} \end{aligned}$$

θ_j^L can be simplified,

$$\begin{aligned} \theta_j^L &= e^{\pi ij} = \cos\pi j + i\sin\pi j \\ &= \begin{cases} 1 & \text{if } j \text{ is even} \\ -1 & \text{if } j \text{ is odd} \end{cases} \end{aligned}$$

And so,

$$\lambda_j = \begin{cases} 0 & \text{if } j \text{ is even} \\ \frac{-4\epsilon}{L(1-2\epsilon)(1-\theta_j)} & \text{if } j \text{ is odd} \end{cases} \tag{4.14}$$

For the information loop to have a stable fixed point at $\rho_{(k)} = 1/(1-2\epsilon)$, we need $|\lambda_j| < 1\ \forall j$. It is only necessary to analyze odd values of j. Recall, $0 < \epsilon < 0.5$.

$$\begin{aligned} |\lambda_j| &= \left| \frac{-4\epsilon}{L(1-2\epsilon)(1-\theta_j)} \right| \\ &= \frac{4\epsilon}{L(1-2\epsilon)|1-\theta_j|} \end{aligned}$$

$|\lambda_j|$ is at its greatest when $|1-\theta_j|$ is at its smallest, i.e. when $j = 1$ or $2L-1$. We will look to see for what values of ϵ, $|\lambda_1| < 1$ holds.

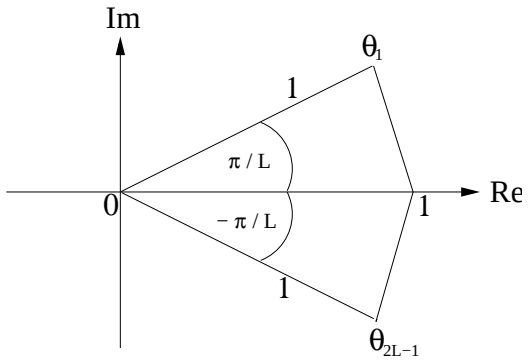

Figure 4.12: Argand diagram showing the distance $|1-\theta_1| = |1-\theta_{2L-1}|$

The distance $|1-\theta_1| = 2\sin(\pi/2L)$ is found using basic trigonometry, figure 4.12. Thus,

$$|\lambda_1| = \frac{2\epsilon}{L(1-2\epsilon)\sin(\pi/2L)}$$

For stability,

$$\begin{aligned}
\frac{2\epsilon}{L(1-2\epsilon)\sin(\pi/2L)} &\leq 1 \\
2\epsilon &\leq L(1-2\epsilon)\sin(\pi/2L) \\
2\epsilon + L2\epsilon\sin(\pi/2L) &\leq L\sin(\pi/2L) \\
\epsilon &\leq \frac{L\sin(\pi/2L)}{2(1+L\sin(\pi/2L))}
\end{aligned}$$

Note that the function $a/(2(1+a))$ is zero at $a = 0$ and rises monotonically towards $1/2$ as $a \to \infty$. So for every value of L there is a value ϵ_* between 0 and $1/2$ such that $\rho_k = 1/(1-2\epsilon)$ is a stable fixed point for $\epsilon < \epsilon_*$, and is unstable for $\epsilon > \epsilon_*$. $L\sin(\pi/2L)$ is a monotonically increasing function of L, so ϵ_* is also monotonically increasing with L.

For small x, $\sin(x) \approx x$, so as L increases,

$$\begin{aligned}
\epsilon_* &= \frac{L\sin(\pi/2L)}{2(1+L\sin(\pi/2L))} \\
&\to \frac{\pi}{2(\pi+2)} \\
&\approx 0.3055
\end{aligned}$$

Hence, for an accuracy information loop the theory suggests that ϵ can be no greater than 0.3055 for there to be a stable fixed point at $\tau_{(k)} = 1-2\epsilon$.

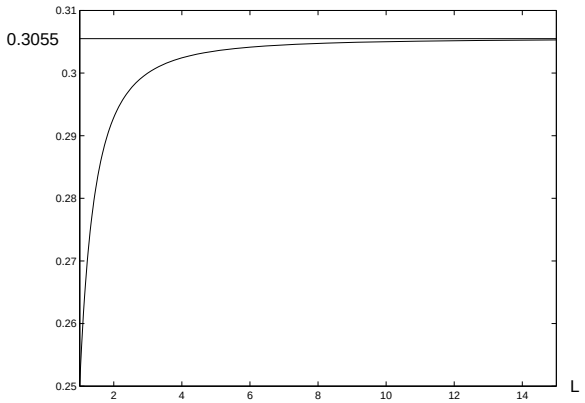

Figure 4.13: Graph of $\epsilon = \frac{L\sin(\pi/2L)}{2(1+L\sin(\pi/2L))}$ and $\epsilon = 0.3055$.

4.2.4 Accuracy information loop simulations

Here we are going to show the results of a couple of accuracy information loop simulations. Results for the no inversion accuracy information loops will not be given, as the loops freeze so quickly. The first simulations show the evolution of $\tau^n_{i,b}$ in the one-inversion information loop. The second concentrate on the accuracy of the nodes in an accuracy information loop.

It is only necessary to observe one value of $\tau^n_{i,b}$ when considering the loops general activity. If this value of $\tau^n_{i,b}$ becomes 0 or $(1-2\epsilon)/(1-\epsilon)$, we know node i has frozen, furthermore all other nodes either, have frozen or will soon freeze. We set $\tau^0_{i,b} = 1 - 2\epsilon \ \forall \ i, b$, and track the value of $\tau^n_{0,0}$ in a loop length 40. A vector memory, length 1000, records the state of its node on every iteration, and is used to estimate $H^n_{i,1}$.

A point to note is that the simulated network does indeed enter the expected limit cycle, and so a node will have output,

$$\ldots, x^{n-1}_i, x^n_i, x^{n+1}_i, x^{n+2}_i, \cdots = \ldots, 0, 0, 0, \ldots, 0, 1, 1, 1, \ldots, 1, 0, 0, \ldots$$

Each block of 0's or 1's will be at least L long ($L = 40$ for our simulation). Thus if the memory is length L the loop is likely to freeze. As we are investigating the fixed point $\tau^n_{i,b} = 1 - 2\epsilon$, we can say that the average length of each block is $L(1-2\epsilon)$. This means that, if we are to get a reliable estimation of $H^n_{i,1}$, the length of our vector memory must be greater than $2L(1-2\epsilon)$.

The memory used to estimate $H^n_{i,1}$ is an empty/fill vector (see section 3.2.2). As in the theory, $G^n_{i,b} = H^n_{i-1,b}$ for the identity nodes, and $G^n_{0,b} = 1 - H^n_{39,b}$ for

the inversion. $\tau^0_{i,b} = \tau^1_{i,b} = \cdots = \tau^{999}_{i,b}$, then $G^{999}_{i,b}$ is deduced using $H^{999}_{i-1,b}$, and the re-evaluation probabilities updated ($\tau^{1000}_{i,b}$). The vector is emptied and the cycle runs again. Figure 4.14 shows 1000 such updates of $\tau^n_{0,0}$, for three values of ϵ; $0.1, 0.2$, and 0.3.

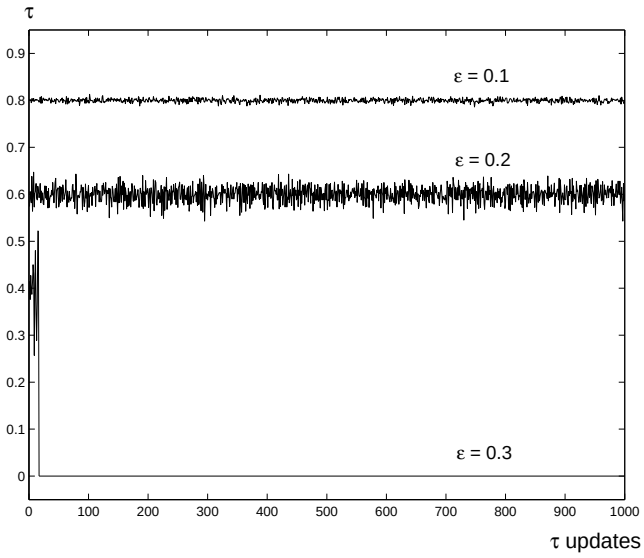

Figure 4.14: The value of $\tau^n_{0,0}$ for 1000 updates.

We can see that for $\epsilon = 0.1$ and 0.2, $\tau^n_{0,0}$ has a mean of $1 - 2\epsilon$. By way of explaining the larger variance on $\tau^n_{0,0}$ when $\epsilon = 0.2$, look at,

$$\begin{aligned} \tau &= 1 - \frac{\epsilon}{1-G} \\ \left.\frac{\partial \tau}{\partial G}\right|_{1/2} &= \left.\frac{-\epsilon}{(1-G)^2}\right|_{1/2} \\ &= -4\epsilon \end{aligned}$$

This shows that any deviation from 0.5 in the estimation of $G^n_{i,b}$ has a greater effect on $\tau^n_{i,b}$ as ϵ increases.

For $\epsilon = 0.3$ the loop freezes, as expected.

The accuracy of an accuracy information loop

It was suspected that the accuracy heuristic would be ineffective on information loops as a result of feedback effects. A number of simulations confirmed this to be the case. The simulations did illuminate an interesting point: accuracy tends to 1 as the loop increases in size even though average $\tau^n_{i,b} = 1-2\epsilon$ (see figure 4.15).

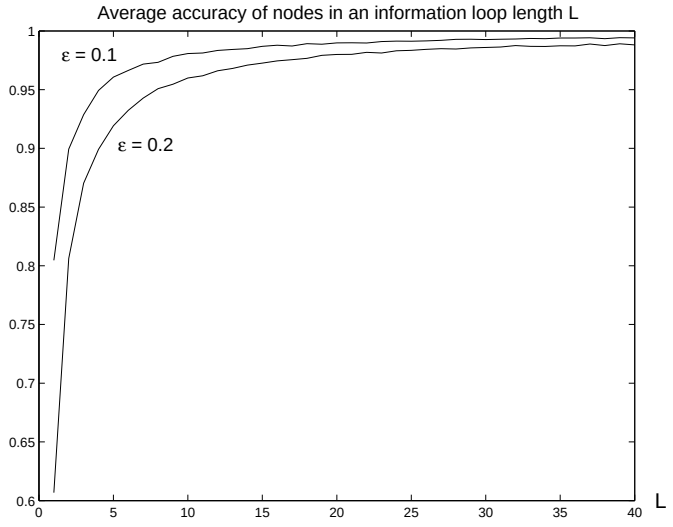

Figure 4.15: The average accuracy of the nodes in information loops, $L = 1, 2, 3, \ldots, 40$, $\epsilon = 0.1, 0.2$.

Figure 4.15 shows the average accuracy for information loops length 1 to 40. The top curve shows the accuracy of the loops for $\epsilon = 0.1$ and the bottom for $\epsilon = 0.2$. For these simulations $\tau^0_{i,b} = 1 - 2\epsilon$, the vector length was 500 and there were 10 updates of $\tau^n_{i,b}$, i.e. each simulation ran for 5000 iterations.

Once the accuracy loop has entered the cycle, each node outputs blocks of zeros and ones. As discussed, the larger L the longer these blocks become. A node can only be inaccurate in the period where it is possible for the block to change from zeros to ones, or vice versa. The larger L, the less frequent that is, and the more accurate the node becomes. For node 0 (for example) this is when the network is in state $000\ldots0$ or $111\ldots1$. Thus node 0 is inaccurate at time n if it is in one of those states at time n and the same state at time $n-1$. This is equivalent to being in the state at $n-1$, and not updating at time n. Hence the probability of being inaccurate is,

$$2\epsilon\mu_{(0)} + 2\epsilon\mu_{(2L-1)} = \frac{2\epsilon}{L}$$

Figure 4.16 shows the accuracy data and the curve $1 - 2\epsilon/L$, for $\epsilon = 0.1, 0.2$.

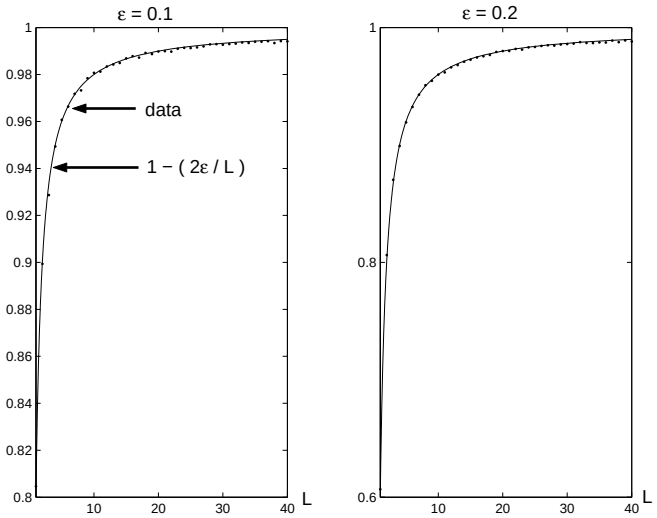

Figure 4.16: Accuracy data, and curve $1 - 2\epsilon/L$

Chapter 5

Random Boolean Networks

In this chapter we will look at the dynamics of deterministic, probabilistic and accuracy RBNs. We will also present results of accuracy simulations and say a few words about the reduction in traffic volume (as a result of the heuristic).

5.1 Dynamics

5.1.1 Distribution of $G(b)$ for classic RBN

The idea behind the accuracy heuristic is that it sits on a deterministic Boolean network (nodes update synchronously on every iteration), to be 'switched on' when required (i.e. initial re-evaluation probabilities are equal to 1). In this subsection we are going to look at the distribution of $G_{i,1}$ across classic RBN's (subsection 1.1.1), which will give us insight into the first non-trivial re-evaluation probabilities (equation (3.6)). Of course the introduction of non-trivial re-evaluation probabilities may alter the distribution of $G_{i,1}$ and thus change the re-evaluation probabilities further; this will be discussed in subsection 5.1.2.

To investigate the existence of a general $G_{i,1}$ distribution across a network with fixed k we set up RBN simulations. Each simulation generated 100 networks with N nodes, with inputs and Boolean functions chosen randomly in the usual way, (subsection 1.1.1). The networks were run for a transient period of 1000 before a vector memory (length 1000) was used to estimate $G_{i,1}$, the hope being that the memory was observing long term behaviour as the dynamics had reached an attractor. We considered networks with 100 and 300 nodes, hence state spaces with 2^{100}, 2^{300} states. A 20 bin histogram of $G_{i,1}$ was constructed for each of the 100 k-networks, and an average histogram was taken. We constructed the 100 histograms separately to calculate the standard deviation of the histograms from their average, to ensure we were not finding a general distribution for $G_{i,1}$ which is

uncharacteristic of most networks with a particular k. RBN simulations were run for $k = 1, 2, 3, 4, 5, 6$ and $N = 100, 300$. Figure 5.1 shows average histograms for $N = 100$, figure 5.2 shows the $G_{i,1}$ histograms for six randomly chosen $N = 100$ networks.

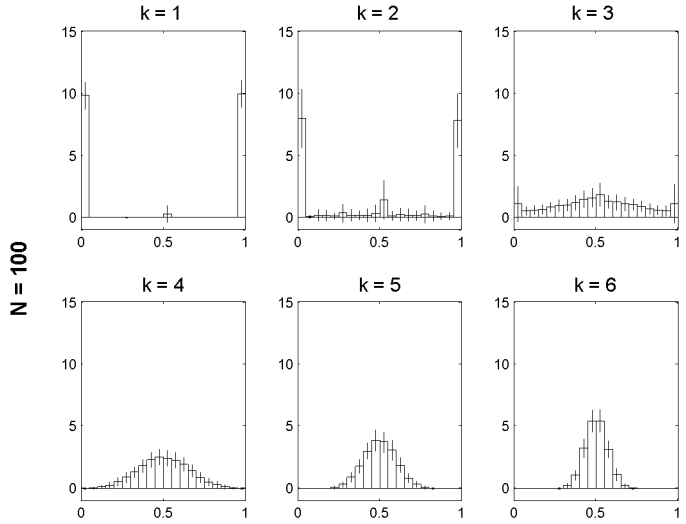

Figure 5.1: The average $G_{i,1}$ distribution and standard deviation, for classic deterministic RBN's simulations, $k = 1, 2, 3, 4, 5, 6$, $N = 100$.

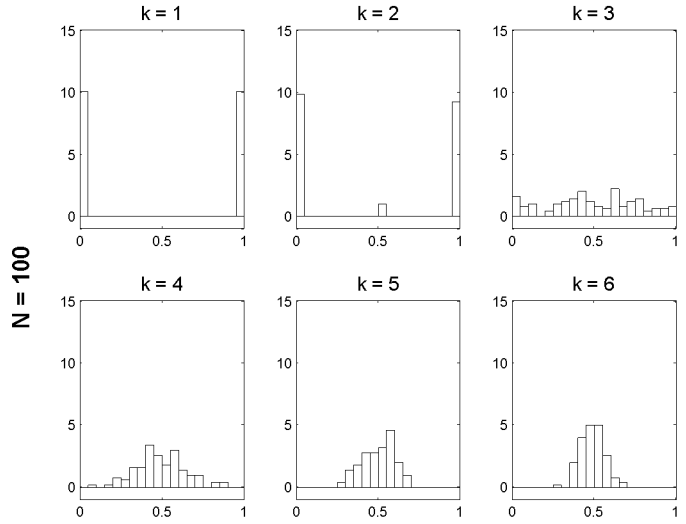

Figure 5.2: The $G_{i,1}$ distribution for one randomly chosen network, $N = 100$, $k = 1, 2, 3, 4, 5, 6$.

As we have stated, we are looking at state spaces with 2^{100}, 2^{300} states, so it may seem that a memory of a node's behaviour for 1000 of those states will not give an

adequate estimation of $G_{i,1}$. In $k = 2$ networks most nodes become frozen greatly reducing the state space (subsection 1.1.2). For $k \geq 3$ most, if not all, nodes are active so the memory is probably not observing a node's complete orbit. However we observe the nodes from a random point on the cycle, and do so for a number of networks (100) thus we consider the distribution of $G_{i,1}$ calculated in this way to be representative.

When comparing the average histograms for $N = 100$ and $N = 300$ (figure 5.1 and 5.3) we see that N does not alter the average distribution of $G_{i,1}$, larger N does however decrease the standard deviation. The standard deviation for both $N = 100$ and 300 is sufficiently small for us to consider these distributions to be describing a general characteristic of classic RBN's for a specified k, (supported by figure 5.2).

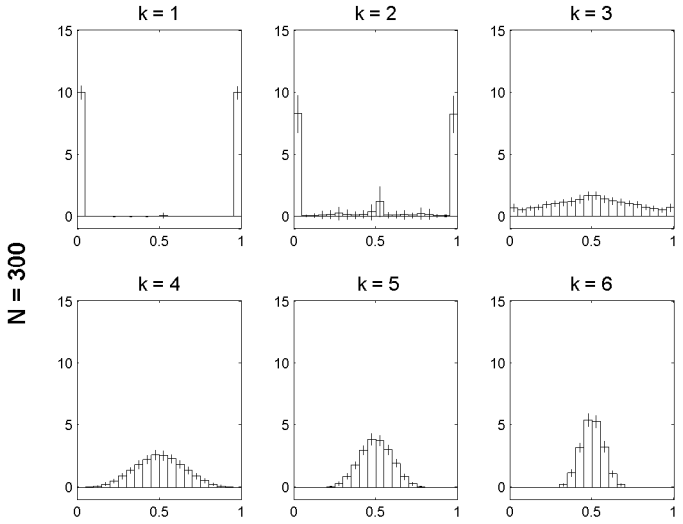

Figure 5.3: The average $G_{i,1}$ distribution and standard deviation, for classic deterministic RBN's simulations, $k = 1, 2, 3, 4, 5, 6$, $N = 300$.

For $k = 4, 5, 6$ the distributions in figures 5.1 and 5.3 look normal. To investigate the normality of RBN $G_{i,1}$ distribution histograms with $k \geq 4$ we fit normal curves to $k = 4, 5, 6, 7, 8, 9, 12, 15$ histograms, $N = 100$. The fitting was done by eye. We varied the number of bins per average histogram to get a clearer indication of its point of inflection, hence a better estimate of the normal standard deviation, σ_G. Table 5.1 shows the estimated values σ_G of for each histogram. Figure 5.4 shows the $G_{i,1}$ histograms and the fitted normal curves.

k	4	5	6	7	8	9	12	15
σ_G	0.155	0.1	0.07	0.048	0.037	0.029	0.018	0.017

Table 5.1: Estimated values of σ_G for each $G_{i,1}$ histogram shown in figure 5.4

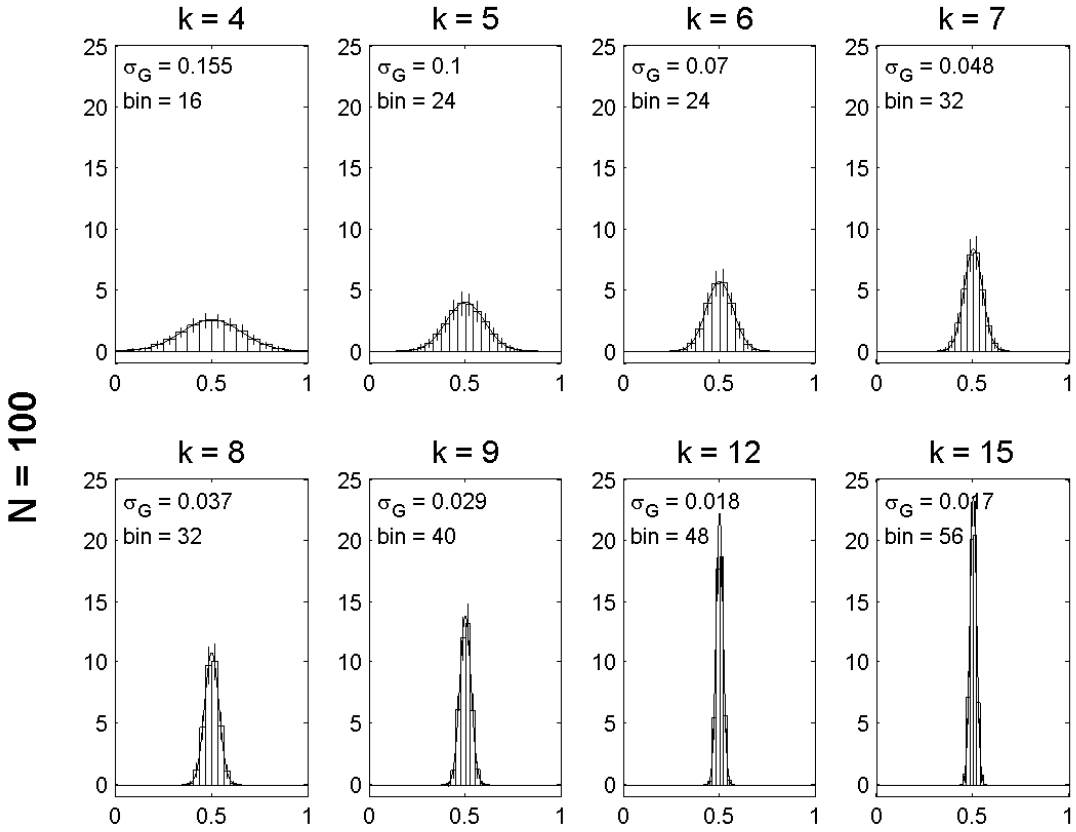

Figure 5.4: The normal fit for average $G_{i,1}$ distribution histograms, $k = 4, 5, 6, 7, 8, 9, 12, 15$, $N = 100$

To find a relationship between σ_G and k, consider a typical node. If all the possible input patterns to the node have equal probability the distribution of the output is controlled by the Boolean function of the node, and in particular $G_{i,1}$ will be the proportion of entries in the truth table equal to one. Since the Boolean functions are chosen at random, the number of 1's in the truth table is binomially distributed, and if this effect dominates the $G_{i,1}$ distributions it could account for their gaussian shape. Below is an example of the distribution of 1's in the

Boolean function truth tables for $k = 2$.

	f^0	f^1	f^2	f^3	f^4	f^5	f^6	f^7	f^8	f^9	f^{10}	f^{11}	f^{12}	f^{13}	f^{14}	f^{15}
00	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
10	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
01	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
11	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
X	0	1	1	2	1	2	2	3	1	2	2	3	2	3	3	4

The random variable X is the number of ones in the truth table of f^j. So we have, $X \sim \mathbf{B}(n, p) = \binom{n}{x} p^x (1-p)^{n-x}$, where p is the probability of on entry being a 1, so $p = 1/2$. For $k = 2$ (above example) $n = 4$ and $X \sim \mathbf{B}(4, 1/2) = \binom{4}{x} 2^{-4}$. More generally, $n = 2^k$, giving the Binomial distribution $X \sim \mathbf{B}(2^k, 1/2) = \binom{2^k}{x} 2^{-2^k}$, with mean $\mu = np = 2^{k-1}$ and standard deviation $\sigma = \sqrt{np(1-p)} = 2^{(k/2)-1}$. For us to compare the Binomial standard deviation to σ_G we need to normalize it so we are looking at the distribution of the proportion of 1's in in the Boolean function truth tables (rather than the number of 1's). Thus, we divide everything by 2^k giving $\sigma_f = 2^{-(k/2)-1}$.

Figure 5.5 shows the result of comparing σ_f with the σ_G (table 5.1). Even though σ_G does not fit the curve σ_f it follows a similar trend. σ_G is always greater than σ_f suggesting there is source of randomness contributing to the spread of $G_{i,1}$ values.

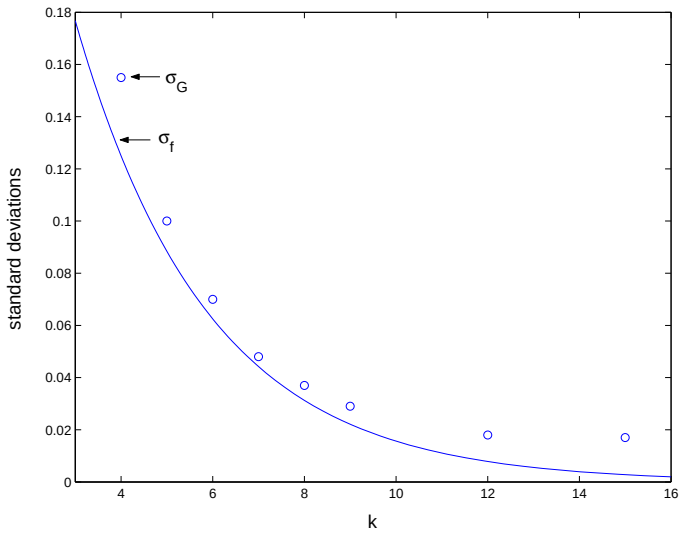

Figure 5.5: A graph showing the estimated σ_G and σ_f

The value of σ_f is the standard deviation of the exact normalised Binomial distribution. Recall the values of $G_{i,1}$ were estimated with a vector length 1000. The fact this length is finite means the estimates are subject to a sampling error. We

shall now derive the normalised Binomial standard deviation for $G_{i,1}$ estimated with a vector length m.

As before, let X be the number of 1's in the truth table of f^j, $p = 1/2$, $n = 2^k$. Now define $P(f^j \text{ outputs a } 1) = p_1 = x/2^k$, m is the length of the vector used to estimate X, and Y be the number of 1's in that vector (so Y/m is our estimate of p_1). We are asking: What would the distribution look like (its mean and standard deviation) if we picked N Boolean functions (with k inputs) at random, and then estimated their p_1 values by using a vector memory and choosing the input vectors at random? Hence, we want to find the variance of normalised Y and take its square root to get,

$$\sigma_{f_m} = \sqrt{\frac{1}{m^2}\sum_{y=0}^{2^k}(y-\mu_Y)^2 P(Y=y)}$$

The probability of having y 1's in the vector depends on p_1 which in turn depends on x. Thus,

$$\begin{aligned} P(Y=y) &= \sum_{x=0}^{2^k} P(Y=y|p_1=x/2^k)P(p_1=x/2^k) \\ &= \sum_{x=0}^{2^k} P(Y=y|p_1=x/2^k)P(X=x) \\ &= \frac{1}{2^{2^k}}\binom{m}{y}\sum_{x=0}^{2^k}\left(\frac{x}{2^k}\right)^y\left(1-\frac{x}{2^k}\right)^{m-y}\binom{2^k}{x} \end{aligned}$$

as both $P(X=x)$ and $P(Y=y|p_1=x/2^k)$ are binomially distributed.

$$\begin{aligned} \mu_Y &= \sum_{y=0}^{m} yP(Y=y) \\ &= \frac{1}{2^{2^k}}\sum_{x=0}^{2^k}\binom{2^k}{x}\sum_{y=0}^{m} y\binom{m}{y}\left(\frac{x}{2^k}\right)^y\left(1-\frac{x}{2^k}\right)^{m-y} \end{aligned}$$

The $\sum_{y=0}^{m}$ term is the mean of the binomial, $Y \sim \mathbf{B}(m, x/2^k)$.

$$\mu_Y = \frac{m}{2^k}\sum_{x=0}^{2^k} x\binom{2^k}{x}\frac{1}{2^{2^k}}$$

Likewise, the $\sum_{x=0}^{2^k}$ term is the mean of the binomial, $X \sim \mathbf{B}(2^k, 1/2)$, giving,

$$\mu_Y = \frac{m}{2}$$

The variance of Y is,

$$\begin{aligned}\text{var } Y &= \sum_{y=0}^{m}\left(y-\frac{m}{2}\right)^2 P(Y=y)\\ &= \frac{1}{2^{2k}}\sum_{x=0}^{2^k}\binom{2^k}{x}\sum_{y=0}^{m}\left(y-\frac{m}{2}\right)^2\binom{m}{y}\left(\frac{x}{2^k}\right)^y\left(1-\frac{x}{2^k}\right)^{m-y} \qquad (5.1)\end{aligned}$$

Note,

$$\begin{aligned}\text{var } \mathbf{B}(m,x/2^k) &= \sum_{y=0}^{m}\left(y-\frac{xm}{2^k}\right)^2\binom{m}{y}\left(\frac{x}{2^k}\right)^y\left(1-\frac{x}{2^k}\right)^{m-y}\\ &= m\frac{x}{2^k}\left(1-\frac{x}{2^k}\right)\end{aligned}$$

and

$$\begin{aligned}\left(y-\frac{m}{2}\right)^2 &= \left(y-\frac{xm}{2^k}+\frac{xm}{2^k}-\frac{m}{2}\right)^2\\ &= \left(y-\frac{xm}{2^k}\right)^2+2m\left(y-\frac{xm}{2^k}\right)\left(\frac{x}{2^k}-\frac{1}{2}\right)+m^2\left(\frac{x}{2^k}-\frac{1}{2}\right)^2\end{aligned} \qquad (5.2)$$

so we can break the sum over m, in equation (5.1), into three parts. One part is simply var $\mathbf{B}(m,x/2^k)$. The expression created by the middle term in equation (5.2) is expanded further to give,

$$\begin{aligned}&2m\left(\frac{x}{2^k}-\frac{1}{2}\right)\left(\sum_{y=0}^{m}y\binom{m}{y}\left(\frac{x}{2^k}\right)^y\left(1-\frac{x}{2^k}\right)^{m-y}-\sum_{y=0}^{m}\frac{xm}{2^k}\binom{m}{y}\left(\frac{x}{2^k}\right)^y\left(1-\frac{x}{2^k}\right)^{m-y}\right)\\ =\ &2m\left(\frac{x}{2^k}-\frac{1}{2}\right)\left(\frac{xm}{2^k}-\frac{xm}{2^k}\right)\\ =\ &0\end{aligned}$$

Finally, we have,

$$\begin{aligned}&m^2\left(\frac{x}{2^k}-\frac{1}{2}\right)^2\sum_{y=0}^{m}\binom{m}{y}\left(\frac{x}{2^k}\right)^y\left(1-\frac{x}{2^k}\right)^{m-y}\\ =\ &m^2\left(\frac{x}{2^k}-\frac{1}{2}\right)^2\end{aligned}$$

Put all this back into equation (5.1) to get,

$$\text{var } Y = m\sum_{x=0}^{2^k}\left(\frac{x}{2^k}\left(1-\frac{x}{2^k}\right)+m\left(\frac{x}{2^k}-\frac{1}{2}\right)^2\right)\binom{2^k}{x}\frac{1}{2^{2^k}}$$

We are going to manipulate this expression to use the property,
var $\mathbf{B}(2^k, 1/2) = \sum_{x=0}^{2^k}(x - 2^{k-1})^2\binom{2^k}{x}2^{-2^k} = 2^{k-1}$

$$\begin{aligned}
\text{var } Y &= \frac{m}{2^{2k}}\sum_{x=0}^{2^k}\left(x2^k - x^2 + m\left(x^2 - x2^k + (2^{k-1})^2\right)\right)\binom{2^k}{x}\frac{1}{2^{2^k}} \\
&= \frac{m(m-1)}{2^{2k}}\sum_{x=0}^{2^k}\left(x^2 - x2^k + \frac{m}{m-1}(2^{k-1})^2\right)\binom{2^k}{x}\frac{1}{2^{2^k}} \\
&= \frac{m(m-1)}{2^{2k}}\sum_{x=0}^{2^k}\left(x^2 - x2^k + (2^{k-1})^2 + \frac{(2^{k-1})^2}{m-1}\right)\binom{2^k}{x}\frac{1}{2^{2^k}} \\
&= \frac{m(m-1)}{2^{2k}}\left(2^{k-2} + \frac{(2^{k-1})^2}{m-1}\right) \\
&= \frac{m}{2^{k+2}}(m - 1 + 2^k)
\end{aligned}$$

We now normalize var Y (dividing by m^2) and take the square root to get,

$$\sigma_{f_m} = \sqrt{\frac{m + 2^k - 1}{m2^{k+2}}} \tag{5.3}$$

Figure 5.6 shows the graph of equation (5.3) for $m = 1000, 6000$. The correlation between σ_G and σ_{f_m} improves as k increases, suggesting that as k is increased the nodes output 0 and 1 with probability 1/2. Note that as expected for increasing m, $\sigma_{f_m} \to \sigma_f$, this can be seen to be true from their equations as well as figure 5.6.

$$\begin{aligned}
\sigma_{f_m} &= \sqrt{\frac{m + 2^k - 1}{m2^{k+2}}} \\
&= \sigma_f\sqrt{1 + \frac{2^k - 1}{m}}
\end{aligned}$$

$$\frac{2^k - 1}{m} \to 0 \quad \text{as} \quad m \to \infty$$

5.1.2 Frozen component

In this subsection we are going to look at what we term the 'frozen component'. The idea behind the frozen component was taken from Flyvbjerg who characterised the dynamics of an RBN by its 'stable core' [7]. For a node to contribute to the stable core its value had to remain fixed for all iterations after some point, independent of initial conditions. Flyvbjerg completed a theoretical analysis of the stable core and concluded that, for $k = 2$ networks the stable core engulfs all

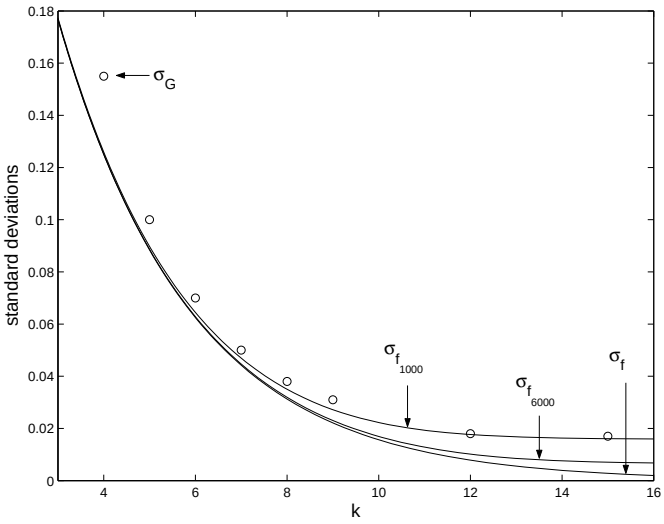

Figure 5.6: A graph comparing the estimated σ_G to σ_f,$\sigma_{f_{1000}}$ and $\sigma_{f_{6000}}$

nodes, for $k \geq 3$ networks it contains hardly any (section 1.1.2).

Investigations into the frozen component are numerical. We define the frozen component as the proportion of nodes that have had a fixed value for a minimum period. The size of the frozen component depends on the minimum period, though if this is long enough (say 2^N for deterministic networks, N being the number of nodes) the frozen component will contain only nodes which are truly frozen that is, nodes whose state is guaranteed not to change for all time. For practical computing, it is likely that the minimum period will be shorter than required (to catch only truly frozen nodes) so unlike the stable core the size of the frozen component can decrease.

The distribution of $G_{i,1}$ over deterministic RBNs gives us information about the first re-evaluation probabilities to have a value less than 1, thus we can make statements about the proportion of nodes that will be frozen immediately by the heuristic (those whose $G_{i,1}$ falls outside the interval $[\epsilon, 1-\epsilon]$). However this tells us nothing about the new $G_{i,1}$ distribution (the $G_{i,1}$ distribution is altered by the onset of non-trivial re-evaluation probabilities), thus we have no information about the new re-evaluation probabilities and if more nodes subsequently become frozen.

In subsection 4.2.2 we discussed how movement through probabilistic state space is guided by movement through deterministic state space. This was illustrated by the example of a state (in state space) $X^n \in \mathbf{B}_0^{12}$ and its deterministic Boolean

map F (consisting of 12 Boolean functions),

$$\begin{array}{ll} X^n & [0\underline{0000}101\underline{11}01] \\ F(X^n) & [100000101110] \end{array}$$

We see that the underlined states in X^n do not change value under the map F so cannot change under probabilistic updating. Hence a probabilistic Boolean map could not map X^n to $1\underline{1000}0101\underline{11}10$, for example. For $k = 2$ deterministic networks once the network has entered its cycle most nodes become frozen (see subsection 1.1.2, and figure 5.1). If we were then to impose probabilistic updating, the only possible way for the frozen nodes to unfreeze is for the network to enter another basin of attraction (i.e. to become a state which leads to another cycle (in deterministic state space) or a state in another cycle). For $k = 2$ networks with probabilistic updating, this is very unlikely as cycles are separated by large basins of attraction. As k is increased the deterministic cycles become fewer and longer with smaller basins. In fact for $k \geq 3$ most if not all nodes in the deterministic network are active, eliminating the question of unfreezing.

The accuracy heuristic has been designed to reduce nodal activity (traffic): a node updates with the minimum probability to assure a certain accuracy, and in some cases the heuristic imposes freezing. We are going to use the frozen component to investigate if and how the freezing imposed by the heuristic spreads across the network. We will show results from three classic RBNs, $k = 2$, $k = 3$, and $k = 5$ (networks 01, 02, and 03 respectively), under two types of probabilistic updating. One set of results will be from the networks running the accuracy heuristic (after a deterministic transient period) for various values of ϵ. The other set of results will be from the same networks running a state independent, probabilistic updating scheme (termed as probabilistic RBNs). During probabilistic RBN simulations a fixed number of nodes, m, are synchronously updated on each iteration. The m nodes are chosen at random on each iteration, giving each node a re-evaluation probability of m/N. Comparing probabilistic RBN results with accuracy RBN results illuminates the dynamical changes due to the enforced freezing of the heuristic. If we were to observe re-evaluation probabilities we would know which nodes were truly frozen (as $\tau_{i,0}$ or $\tau_{i,1}$ would be equal to zero) but this method does not encompass probabilistic RBNs, and so we chose to use the minimum period.

Network simulations had initial re-evaluation probabilities $\tau^0_{i,0} = \tau^0_{i,1} = 1$, allowing us to record the frozen component of the deterministic dynamics before probabilistic updating occurred. No frozen component measurements were taken

before a transient period of 7000 iterations (so network was on a cycle). For accuracy networks $G_{i,1}$ was estimated at the end of the deterministic phase (directly with scalar memory, $\omega = 0.998$ (see subsection 3.2.1)) and used to give the first non-deterministic re-evaluation probabilities. The minimum period during simulations was 1000.

For $k = 2$ RBNs we expect most of the nodes to be frozen and a small number to have $G_{i,1}$ close to 0.5 (figure 5.1), and so the accuracy heuristic will not initially freeze any extra nodes unless $\epsilon \approx 0.5$. In subsection 4.2.3 we saw that for odd inversion accuracy information loops $G_{i,b} = 0.5$ was a stable fixed point of the heuristic.

If most of the nodes in a $k = 2$ network are frozen, one of the inputs to a non-frozen node is likely to be frozen, so the node is effectively $k = 1$, and the active nodes form information loops. Figure 5.7 shows the active sub-network in network 01, $G_{i,b} \approx 0.5$ for nodes $4, 9, 15.94$ and 95. This does suggest that for large enough ϵ freezing will occur, but the active component seems more stable in figure 5.8 than we would expect on this basis (subsection 4.2.3). If it were unstable the introduction of re-evaluation probabilities would alter $G_{i,1}$ enough to force it out of $[\epsilon, 1 - \epsilon]$, and freeze the nodes.

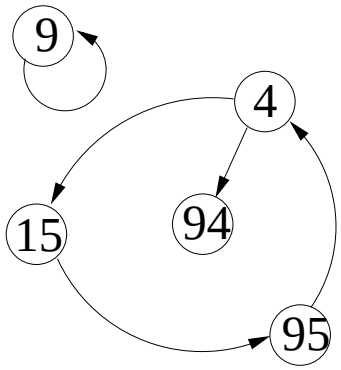

Figure 5.7: Active sub-network of network 01.

The top graph in figure 5.8 shows the frozen component of network 01 (a $k = 2$ RBN with 100 nodes) running the accuracy heuristic. The deterministic frozen component is 0.95, this remains unchanged for $\epsilon < 0.49$ suggesting $G_{i,b} \approx 0.5$ is a stable fixed point of the heuristic for this network. One can think of probabilistic updating as subjecting the deterministic network to noise (a method used by Kauffman to test stability [11]. However, Kauffman flipped bits at random so

the noise he subjected networks to could free dynamics from a fixed point, unlike probabilistic networks. Map (1.1) has an unstable fixed point). The accuracy RBN simulation suggests that the deterministic cycle (ϵ=0) is stable (resilient to noise) as the frozen component remains unchanged. This is supported by the frozen component results for the same network running probabilistic updating (bottom graph in figure 5.8).

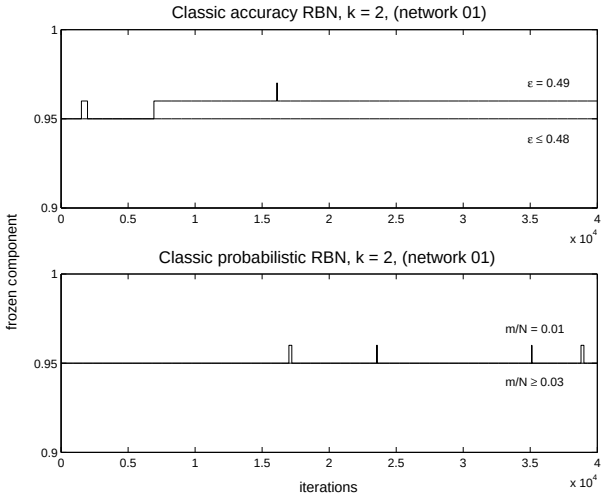

Figure 5.8: Frozen component graphs for network 01 (a $k = 2$ RBN with 100 nodes) running the accuracy heuristic (top), and probabilistic updating (bottom).

The $G_{i,1}$ distribution for $k = 3$ (figure 5.1) shows values of $G_{i,1}$ across the unit interval, so all values of ϵ have the potential to freeze nodes. As k is increased further the interval supporting $G_{i,1}$ shrinks until $G_{i,1} \approx 0.5$ for all nodes (figure 5.4) reducing the chance of freezing by smaller values of ϵ. Figures 5.9 and 5.10 show the frozen components of a $k = 3$ and $k = 5$ RBN, (networks 02 and 03 respectively). Looking first at the accuracy networks: the onset of freezing in network 03 is at a higher value of ϵ than that of network 02, as expected. However, their frozen components evolve in a similar manner: for most values of ϵ (which promote freezing) there is a gradual increase in the frozen component before a levelling off. The major difference in the two networks is network 02 has a fixed point that can be reached from the perturbed deterministic cycle. The values of ϵ (in network 02) that have not resulted in the network becoming frozen may have done if the simulation was run for longer. Another reason why these values of ϵ have not caused the network to freeze is that the accuracy heuristic may have frozen a node to a value which is not in the fixed point. We see from the probabilistic frozen component of network 02 (right graph in figure 5.9) that the

more noise (the smaller the value of m/N) we subject the network to, the sooner it finds the fixed point.

When Kauffman did his stability experiments he only perturbed one node at a time, equivalent to $m/N = 0.99$ in our experiments: for this value of m/N our stable component remains constant. In network 01 we assumed that if the frozen component remained constant (for increasing ϵ, decreasing m) the network had not left the deterministic cycle. We know from the literature (subsection 1.1.2) that $k = 2$ RBNs have small cycles separated by large basins of attraction. It is unlikely that the number of active nodes in these cycles are the same, thus we would expect the size of the frozen component to change if the networked moved from one (deterministic) cycle to another. It becomes more difficult to say if a constant frozen component indicates the network is in the same (deterministic) cycle for larger k as the cycles become longer forcing most if not all nodes to be active, i.e. cycles usually have the same deterministic frozen component of zero.

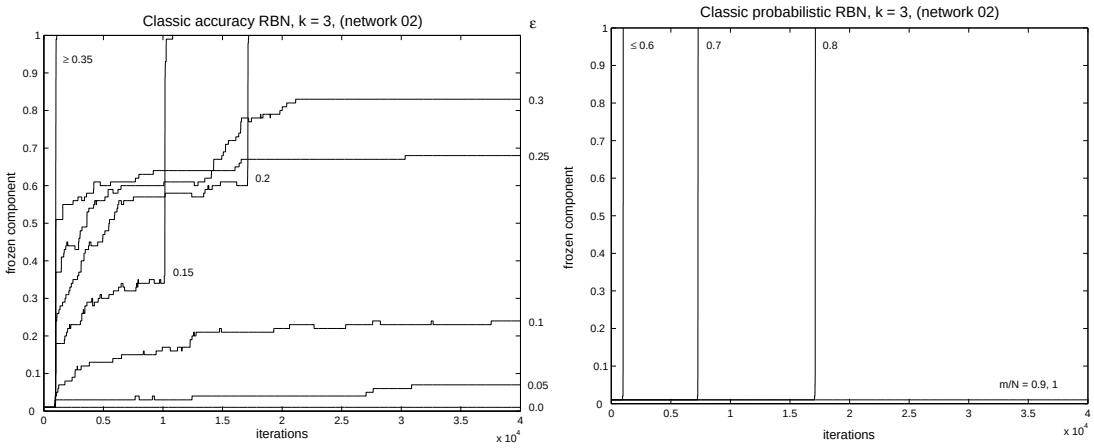

Figure 5.9: Frozen component graphs for network 02 (a $k = 3$ RBN with 100 nodes) running the accuracy heuristic (left), and probabilistic updating (right).

If the value of the frozen component stabilises for a given ϵ it is possible that the network has converged (i.e. $\tau_{i,b}$'s have converged) and $G_{i,b}$'s are stationary. This is certainly not true if the frozen component continues to increase. We see from frozen component graphs that the value of the frozen component can level off, but can also have constant periods before periods of increase. In fact there is always a finite probability of nodes (in an accuracy RBN) freezing because of the probabilistic nature of updates and the finite memory estimation of $G_{i,b}$. As for the possible lack of stationarity, the heuristic can react to a change in $G_{i,b}$ but the

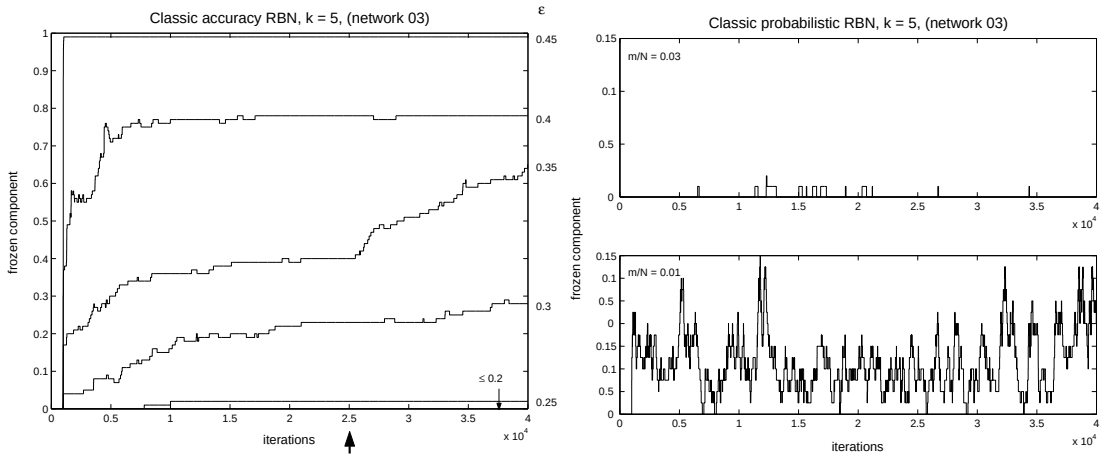

Figure 5.10: Frozen component graphs for network 03 (a $k = 5$ RBN with 100 nodes) running the accuracy heuristic (left), and probabilistic updating (right).

reaction is delayed by the number of iterations needed to correct the estimation of the distribution.

The general picture suggested by the simulations is that for $k = 2$ the accuracy networks, like their deterministic counterparts, have large frozen components. The size of the frozen component can increase from the deterministic value, but only for relatively high values of ϵ. For $k \geq 3$, the accuracy networks can develop significant frozen components, their size increasing with ϵ. As k increases, the value of ϵ needed to produce a frozen component of a given size also increases.

Frozen components of general RBNs behave very similarly to those of classic RBNs. The in-degree (k_i) of nodes in general RBNs are chosen randomly from the interval $[1, k_{max}]$ so there are, on average, N/k_{max} nodes with in-degree 1, N/k_{max} nodes with in-degree 2 etc. General RBNs with average in-degree $\langle k \rangle = 2$ behave similarly to $k = 2$ classic RBNs: they have short cycles with large basins of attraction. As $\langle k \rangle$ is increased the cycles become longer and basins smaller. So, the frozen component similarity between general and classic RBNs with $\langle k \rangle = k$ it is not such a surprise. The major difference when comparing general RBNs is the value of their deterministic frozen components: for $\langle k \rangle = 2$ networks it is smaller due to nodes with in-degree greater than 2, for $\langle k \rangle \geq 3$ it is larger due to nodes with in-degree 1, 2.

Frozen component results show there is a qualitative difference in the dynam-

ical behaviour of RBNs when $k = 2$ and $k \geq 3$, with probabilistic updating (either state dependent or independent), which stands up to reason: though it is possible for probabilistic RBNs to move into basins with greater activity, it is highly unlikely, thus it is rare for probabilistic updating to increase the effective size of the state space. As a result trajectories in probabilistic $k = 2$ state space cannot move far from each other (distance measured using the Hamming distance), regardless of their non-deterministic property. Trajectories in probabilistic $k \geq 3$ state space, like their deterministic equivalent, have the ability to jump from one basin to another making it highly probable that they will become separated by a reasonable distance.

5.1.3 Annealed approximation

Let $X \in \{0,1\}^N$ be a state in the state space of a RBN with N nodes. Denote the state of node i as x_i, thus $X = x_0 x_1 x_2 \dots x_{N-1}$. A method for gauging the stability of RBN dynamics is to take a state in a cycle X, flip one of the node states $(x_i \oplus 1)$ giving a new state $\tilde{X}$, then see if $\tilde{X}$ is in the same basin of attraction as X. If the perturbed state $(\tilde{X})$ is in the same basin of attraction as X then we may have a stable cycle. Numerous states in the cycle are chosen randomly and perturbed (the perturbed node is also chosen randomly): if all or most of the perturbed states are in the basin of attraction then the cycle is considered to be stable. It is well documented that for $k = 2$ RBNs most cycles are stable, for $k = 3$ most cycles are unstable, and as k is increased the likelihood of finding a stable cycle is dramatically decreased (subsection 1.1.2).

Let d_n denote the normalised Hamming distance between the two states X^n and $\tilde{X}^n$ at time n,

$$d_n = \frac{1}{N} \sum_{i=0}^{N-1} |x_i^n - \tilde{x}_i^n|$$

Derrida and Pomeau derived an expression for d_{n+1} as a function of d_n by averaging over many boolean networks with fixed k, N, but random architecture. In other words they considered d_{n+1} of a Boolean network with fixed N, k, but the links and Boolean functions were randomly assigned on every iteration (discussed in full in subsection 1.1.2 [6]). Their expression is,

$$d_{n+1} \quad = \quad \frac{1}{2}(1 - (1 - d_n)^k) \tag{5.4}$$

Figure 5.11 shows the graph of Derrida and Pomeau's equation, equation (5.4), for $k = 1, 2, 3, 4$, and $d_n = d_{n+1}$. From graphical analysis the fixed point at the

origin is stable for $k \leq 2$ and unstable otherwise, confirmed by the gradient of equation (5.4) at the origin.

$$\left.\frac{d}{dd_n} d_{n+1}\right|_{d_n=0} = \frac{k}{2}$$

The gradient is 1 at the point where this fixed point changes from stable to unstable and so $k = 2$ is a critical value: for $k \leq 2$ close trajectories (i.e. with initial points $d_n << 1$ apart) converge, $k > 2$ close trajectories diverge (they are drawn to the other fixed point $d_n = d_{n+1} = 1 - (2/k)^{(k-1)/2}$).

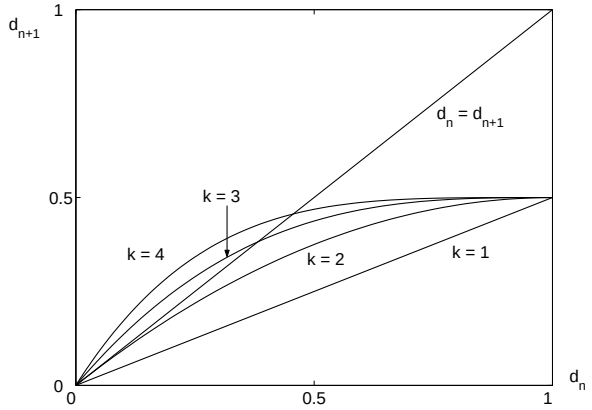

Figure 5.11: Graph of equation (5.4) for $k = 1, 2, 3, 4$, and $d_n = d_{n+1}$.

Mesot and Teuscher go on to develop Derrida and Pomeau's annealed approximation for semi-synchronous, probabilistic RBNs [17]. The updating scheme updates m nodes synchronously on every iteration where m is chosen randomly in the range $[1, N]$; given m, the expected (state independent) update probability for a node is m/N, and the overall probability of being updated is,

$$P_u = \frac{1}{N} \sum_{m=1}^{N} \frac{m}{N} = \frac{N+1}{2N}$$

As the size of the network increases $P_u \to 1/2$.

The limit cycle attractors of the deterministic RBNs no longer occur in the semi-synchronous probabilistic RBNs, but we may still regard the evolution of d_n of two nearby states as giving some indication of stability. Mesot and Teuscher attempt to give an approximate description of this evolution using similar ideas to those of Derrida and Pomeau, (who considered only deterministic RBNs). Mesot and Teuscher's annealed approximation is based on the following procedure:

Method I

(IA) Fix N, k

(IB) Generate a RBN

(IC) Choose X^n and $\tilde{X}^n$ distance d_n apart

(ID) Choose and fix m

(IE) Choose m nodes for updating, and generate X^{n+1}

(IF) Choose another m nodes for updating, and generate $\tilde{X}^{n+1}$

(IG) Calculate d_{n+1}

Mesot and Teuscher choose m randomly on each iteration, (ID). It is not clear why, if they allow the update nodes to be different in the two networks, why they do not allow m to be different.

Let; P_0 be the probability of node i being updated in neither (IE) or (IF), P_1 be the probability of node i being updated in (IE) or (IF) but not both, and P_2 be the probability of node i being updated in both (IE) and (IF). $P_u = 1/2$ (as we are dealing with very large networks) giving, $P_0 = 1/4$, $P_1 = 1/2$ and $P_2 = 1/4$.

Denote the set of nodes for which $x_i^n = \tilde{x}_i^n$ as A, and the set containing all other nodes as set B. Let a_n be the probability that a node chosen at random is in set A; note that $a_n = 1 - d_n$. If a node has all its inputs in set A at n then the probability of that node being in set A at $n+1$ is P_α,

$$P_\alpha = a_n^k \left(P_2 + \frac{P_1}{2} + a_n P_0 \right)$$

P_1 is multiplied by $1/2$ because of the annealed approximation, (there is a probability $1/2$ of node i being in A if it is only updated in one of the states in X^n or $\tilde{X}^n$ because the links and Boolean functions are re-assigned on every iteration). P_0 is multiplied by a_n as the node must be in A at n if it is to be in A at $n+1$, with no update. Thus,

$$P_\alpha = a_n^k \left(\frac{1}{2} + \frac{a_n}{4} \right)$$

If a node does not have all its inputs in A then the probability of it being in set A at $n+1$ is P_β,

$$\begin{aligned} P_\beta &= (1 - a_n^k) \left(\frac{P_2}{2} + \frac{P_1}{2} + a_n P_0 \right) \\ &= (1 - a_n^k) \left(\frac{3}{8} + \frac{a_n}{4} \right) \end{aligned}$$

And so,

$$\begin{aligned} a_{n+1} &= P_\alpha + P_\beta \\ &= \frac{1}{8}(a_n^k + 2a_n + 3) \end{aligned}$$

Rewriting this in terms of the Hamming distance we get,

$$d_{n+1} = \frac{1}{8}(5 - 2(1 - d_n) - (1 - d_n)^k) \tag{5.5}$$

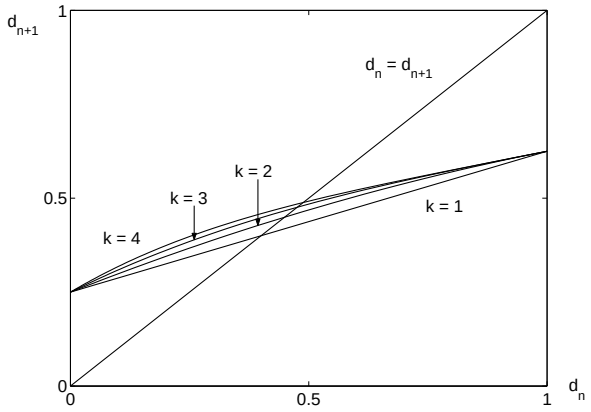

Figure 5.12: The graph of equation (5.5) for $k = 1, 2, 3, 4$, and $d_n = d_{n+1}$.

Figure 5.12 shows the graph of Mesot and Teuscher's annealed approximation (equation (5.5)) for $k = 1, 2, 3, 4$.

One way to think about the difference between deterministic RBNs and probabilistic ones is to regard the probabilistic networks as systems with inputs. For probabilistic networks it is no longer sufficient to know the state of the network to be able to deduce the next state; further information must be supplied, which constitutes an input to the system. For the semi-synchronous (or accuracy) RBNs, this extra information is the identities of the nodes that update - so the input in these cases is random.

We discussed above the idea that the stability of a system relates to how the behaviour is affected if a small change is made to the initial state (we say the system is stable if these behaviours are similar). For autonomous systems (without input) this is simply a matter of comparing the evolutions of two distinct (but nearby) states. For systems with input there are two situations; in both we again compare the behaviour of nearby states, but in the first case we say the

subsequent evolutions are governed by the same inputs, whereas in the second we allow the states to be subjected to different inputs.

Both of these give rise to notions of stability. This notion is clearly stronger in the second case than in the first. In the second we are saying that the system is stable if nearby states have similar behaviours irrespective if input, thus in such systems the input is (or eventually becomes) irrelevant. This is clearly the sense that Mesot and Teuscher are using, because they subject their different initial conditions to different inputs. Their conclusion is that semi-synchronous RBNs are not stable in this sense. This is hardly surprising since we would not expect the random input to be irrelevant in these networks (although one way it could become so would be if there were a fixed point which could capture trajectories).

Less obvious is how the stability of the first sense varies with k. Here we apply the annealing idea to this case.
Method II

(IIA) Fix N, k
(IIB) Generate a RBN
(IIC) Choose X^n and $\tilde{X}^n$ distance d_n apart
(IID) Choose and fix m
(IIE) Choose m nodes for updating, and generate X^{n+1}, $\tilde{X}^{n+1}$
(IIF) Calculate d_{n+1}

For method II there are only two ways to produce x_i^{n+1}, $\tilde{x}_i^{n+1}$ from x_i^n, $\tilde{x}_i^n$: either node i updates or it does not, both events occur with equal probability ($P_u = (1 - P_u) = 1/2$ for $N \to \infty$). As before we look at nodes with all inputs in set A, and nodes with at least one input in set B, separately. If a node has all its inputs in set A it will be in set A after one iteration if: it updates, or it does not update and is itself in A. If a node has at least one input in set B at n, then it will be in set A at $n+1$ if: it does not update and is itself in A at n, or it does update thus will be in set A with the probability $1/2$ (because of the annealed approximation). So,

$$\begin{aligned} a_{n+1} &= a_n^k \left(\frac{1}{2}(1 + a_n) \right) + (1 - a_n^k) \frac{1}{4} (2a_n + 1) \\ &= \frac{1}{4}(a_n^k + 2a_n + 1) \end{aligned}$$

giving the distance equation,

$$d_{n+1} = \frac{1}{4}(3 - 2(1 - d_n) - (1 - d_n)^k) \tag{5.6}$$

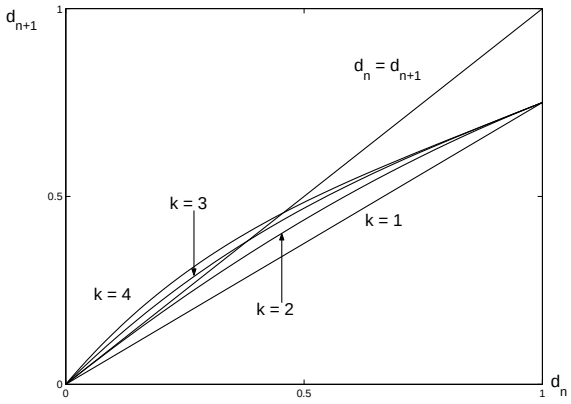

Figure 5.13: The graph of equation (5.6) for $k = 1, 2, 3, 4$ and $d_n = d_{n+1}$

We see that equation (5.6) has a fixed point at $d_n = d_{n+1} = 0$ which changes stability at $k = 2$.

$$\frac{d}{dd_n} d_{n+1}\bigg|_{d_n=0} = \frac{2+k}{4}$$

When the gradient at the origin equals 1, $(2 + k)/4 = 1 \Rightarrow k = 2$.

Equation (5.6) gives the same critical value for k as equation (5.4). This is supported by the numerical experiments in the previous subsection (see figures 5.8, 5.9 and 5.10) with the survival of the frozen component in the $k = 2$ network.

Rather than choosing m at random each iteration and using $P_u = 1/2$ in the derivation of equation (5.6) we could fix m and use $P = m/N$ to see how the choice of m effects the convergence to the fixed points. And so we get,

$$a_{n+1} = \frac{1}{2}\left(a_n^k P + 2a_n(1-P) + P\right)$$

thus,

$$d_{n+1} = 1 - (1-P)(1-d_n) - \frac{P}{2}(1 + (1-d_n)^k) \tag{5.7}$$

Curves of equation (5.7) are shown in figure 5.14 for $P = 1$ (deterministic, Derrida plot) and $P = 1/2$ ($m = N/2$). There are two points on each curve independent of k, $(0, 0)$ and $(1, 1 - P/2)$. We see that, as one might expect, the higher m the quicker d_n converges to its fixed points.

We will extend our annealed approximation (equation (5.6)) to general RBNs.

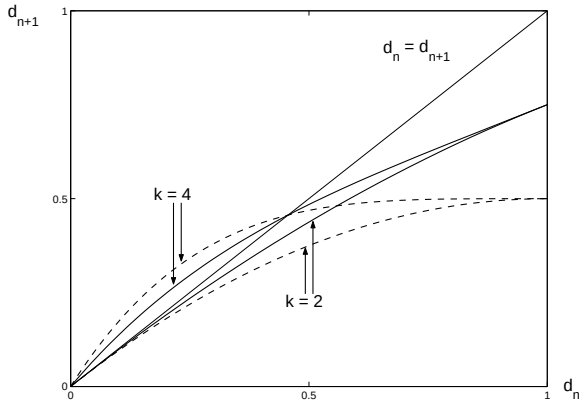

Figure 5.14: The solid lines show equation (5.7) for $P = 1/2$, the dashed lines show equation (5.7) for $P = 1$ (deterministic).

For a general RBN each node has $k_i \in [1, k_{max}]$ inputs. Let P_{k_i} be the distribution of k_i, so the expectation of k_i is,

$$\langle k_i \rangle = \sum_{k_i=1}^{k_{max}} P_{k_i} k_i$$

Using an argument similar to the one that led to equation (5.6) we get,

$$d_{n+1} = \frac{1}{4}\left(3 - 2(1 - d_n) - \sum_{k_i=1}^{k_{max}} P_{k_i}(1 - d_n)^{k_i}\right) \tag{5.8}$$

One way to determine k_i is to choose it randomly from the interval $[1, k_{max}]$, making $P_{k_i} = 1/k_{max}$ and $\langle k_i \rangle = (1 + k_{max})/2$. Figure 5.15 shows equation (5.8) for $\langle k_i \rangle = 2, 3, 4, 5$ ($k_{max} = 1, 3, 5, 7$ respectively). The curves are similar to those in figure 5.13. The slope at the origin is,

$$\frac{d}{dd_n} d_{n+1} = \frac{2 + \langle k_i \rangle}{4}$$

so there is a critical value at $\langle k_i \rangle = 2$, as in the deterministic case.

Annealed approximation for accuracy RBNs

The accuracy RBN is a semi-synchronous, state dependent, probabilistic RBN. A node's re-evaluation probabilities ($\tau_{i,b}$) depend on its $G_{i,b}$ distribution (equation (3.6)). In the annealed approximation of accuracy RBNs $G_{i,b} = 1/2$ so re-evaluation probabilities become state independent and equal to $\tau = 1 - 2\epsilon$, $\epsilon \in [0, 0.5]$. We see that the annealed approximation for accuracy RBNs is similar to that of the the probabilistic RBN (we can set $P = 1 - 2\epsilon$, equation (5.7)),

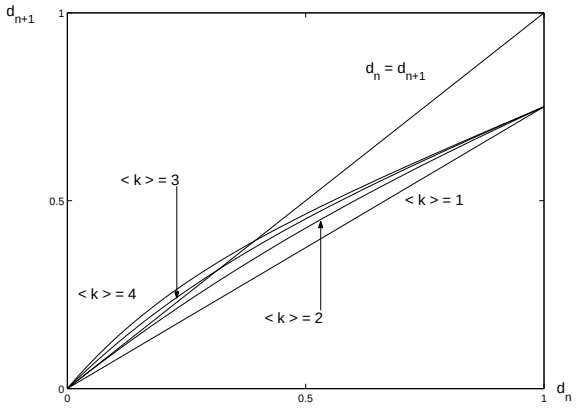

Figure 5.15: Equation (5.8) for $\langle k_i \rangle = 2, 3, 4, 5$ ($k_{max} = 1, 3, 5, 7$ respectively)

since the argument leading to this equation does not use the fact that $P = m/N$ for integers m, N.

The annealed approximation reduces accuracy RBNs to probabilistic RBNs as the continual reassignment of links and Boolean functions eliminates long term behaviour and forces $G_{i,b} = 1/2$. Thus the approximation can not show the effects of enforced freezing (recall a node is frozen if $G_{i,b}$ falls outside the interval $[\epsilon, 1 - \epsilon]$). So although the annealed approximation indicates a change of behaviour at $k = 2$ as usual, it does not capture the smoother change over from frozen (at $k = 2$) to active (for increasing k), seen in subsection 5.1.2.

5.2 Accuracy

Within this section we are going to look at the accuracy of nodes in accuracy RBN's. In deriving the accuracy heuristic (equation (3.6)) we assume $G_{i,b}$ to be iid. From our work on accuracy Boolean trees (subsection 4.1) we know that $G_{i,b}$ is not independent but is positively correlated (at lag 1 at least). This tends to increase the accuracy of a node: a node with positively correlated input is more likely to be accurate (when it has not updated) than it would be if the input were i.i.d. It does however mean there is scope for reducing the re-evaluation probabilities further (hence reducing traffic); this idea is not developed here. In subsection 5.1.2 we observed the frozen component of accuracy RBNs. Frozen component analysis suggested that the behaviour of a network was more likely to stabilize for smaller ϵ (making $G_{i,b}$ stationary). As we discussed, the heuristic can accommodate changes in $G_{i,b}$, with a reaction time depending on the weight parameter for the scalar estimator, or the length of the vector for the vector es-

timator (subsection 3.2.2).

We will show the accuracy of nodes in networks 01 and 03 (the $k = 2$ and $k = 5$ classic RBNs discussed in subsection 5.1.2) calculated by an external observer i.e. one who can see all the node states, or at least the states of the node and its predecessors. For network 01 there are only 5 of the 100 nodes active for most values of ϵ. We observe nodal accuracy for $\epsilon = 0.1$ i.e. we are looking for an accuracy of 0.9. $G_{i,1}$ is estimated directly using the scalar memory, $\omega = 0.998$.

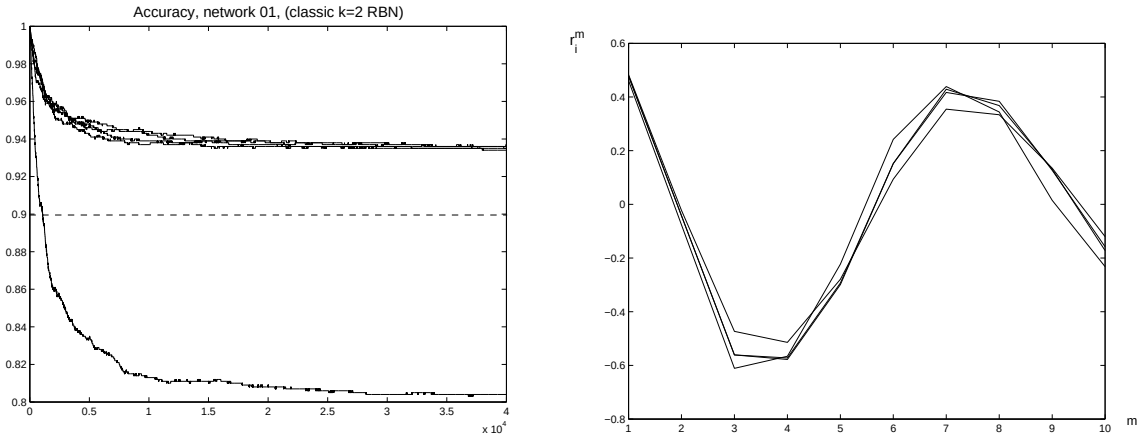

Figure 5.16: Left: The accuracy of active nodes in network 01 (a $k = 2$ RBN), $\epsilon = 0.1$. Right: Autocorrelation of $\{X_i^n\}$ for active (non-self inputting) nodes in the same network.

We see in figure 5.16 that 4 of the 5 active nodes have an accuracy around 0.935. The high accuracy is expected because of the positive correlation in the $\{X_i^n\}$ (the bit string used to estimate $G_{i,b}$). The low accuracy reading is given by node 9 which is a self inputting node (the other input is frozen, see figure 5.7) with $G_{i,1} = 0.5$ hence $\tau_{i,b} = 1 - 2\epsilon = 0.8$, this explains its accuracy, since to be active its Boolean function (with one input frozen) must be the inversion, so it can only be accurate when it updates. The autocorrelations, (r_i^m, for lag $m = 1, 2, \ldots, 10$, equation (4.1)) of the four connected nodes are shown in figure 5.16. The nodes have approximately the same autocorrelation at all lags, and the autocorrelation at lag 1 is more twice that for a node in a tree with one probabilistic updating input (figure 4.5), both these facts can be attributed to the loop topology. This high correlation explains the excessive accuracy of the nodes (note that the self-inputting node 9 has negative correlation at lag 1).

For network 03, $\epsilon = 0.1$, no nodes become frozen. The accuracies of the nodes are shown in figure 5.17 (left). The average accuracy is closer to 0.9 (ranging from 0.9 to 0.915) than the active, non-self inputting, nodes in network 01, this is due to the increased in-degree of each node and hence the reduction in autocorrelation (figure 5.17, right). RBNs contain feedback loops that will increase the correlation in $\{X_i^n\}$, and so the autocorrelations at lag 1, for the 2 randomly chosen network 03 nodes, are much higher than that shown in figure 4.5 (node with probabilistic inputs, in a tree topology). Feedback effects decrease as the number of nodes in a RBN increase, thus the correlation of $\{X_i^n\}$ (for nodes in a RBN) will approach the values shown for nodes in a tree as N increases.

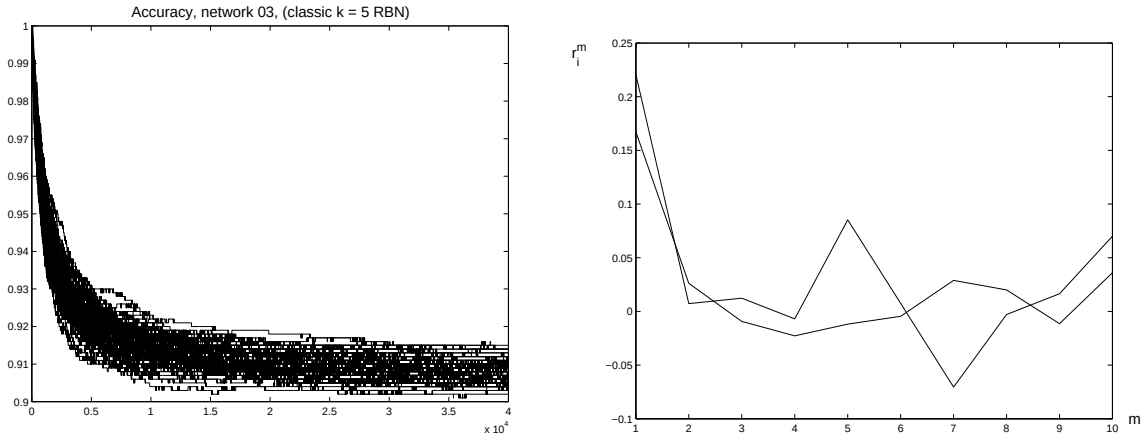

Figure 5.17: Left: The accuracy of nodes in network 03 (a $k = 5$ RBN, all nodes remain active), $\epsilon = 0.1$. Right: Autocorrelation of $\{X_i^n\}$ for two randomly chosen nodes in the same network.

Figure 5.18 shows the average accuracy of active (non-self inputting) nodes in RBNs with $k = 2, 3, 5$ and 9. The average accuracy was taken for 3 classic RBNs, for each in-degree, and then an average (over the 3 networks) was taken and plotted against time. We can see that as k increases (and the autocorrelation at lag 1 decreases) the actual accuracy approaches the desired accuracy.

Note that as well as k, and N, the autocorrelation depends on ϵ; the larger ϵ the smaller the re-evaluation probabilities and the larger the positive correlation.

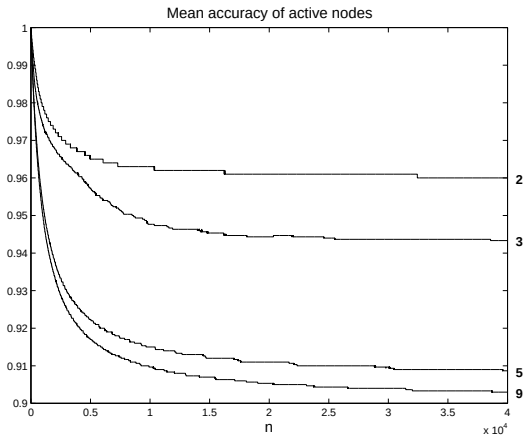

Figure 5.18: The average accuracy of active (non-self inputting) nodes in RBNs with $k = 2, 3, 5$ and 9, $\epsilon = 0.1$

5.3 Traffic reduction

The motivation behind the accuracy heuristic was traffic reduction in computer networks (section 2.1). In this section we are going to give a little thought to the effect of the heuristic on traffic in our abstracted information networks (RBNs).

A probabilistically updating Boolean network can be thought of as a push-pull network, that is a node requests information from its predecessors when it wants to update, and the predecessors send the information in the form of a reply. Call one unit of traffic a request for information and the reply, i.e. whenever a node updates it generates k units of traffic, (or k_i if the node is in a general RBN). We will go on to talk about normalised traffic volume, ν, at time n, defined as the total units of traffic divided by Nk (N = number of nodes in the network). If all nodes are updating synchronously on each iteration (we have previously called this deterministic updating) the network will have a normalised traffic volume of 1.

Assuming node i has been implementing the accuracy heuristic the probability of updating is (equation (3.22)):

$$P_{u_i} = \frac{\tau_{i,0}\tau_{i,1}}{G_{i,1}(\tau_{i,0} - \tau_{i,1}) + \tau_{i,1}}$$

Rewriting P_{u_i} with the substitution (equation 3.6),

$$\tau_{i,1} = \begin{cases} 1 - \frac{\epsilon}{1-G_{i,1}} & \text{for } 0 \leq G_{i,1} \leq 1-\epsilon \\ 0 & \text{otherwise} \end{cases}$$

$$\tau_{i,0} = \begin{cases} 1 - \frac{\epsilon}{G_{i,1}} & \text{for } \epsilon \leq G_{i,1} \leq 1 \\ 0 & \text{otherwise} \end{cases}$$

gives,

$$P_{u_i} = \begin{cases} \frac{1}{1-2\epsilon}\left(1 - \frac{\epsilon(1-\epsilon)}{G_{i,1}(1-G_{i,1})}\right) & \text{for } \epsilon \leq G_{i,1} \leq 1-\epsilon \\ 0 & \text{otherwise} \end{cases} \tag{5.9}$$

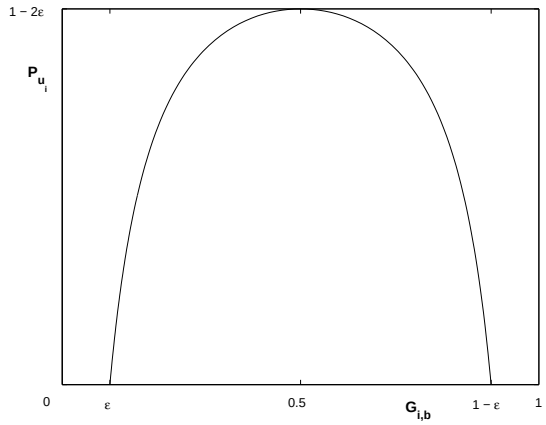

Figure 5.19: The graph of P_{u_i}, equation (5.9).

We can see from figure 5.19 that the probability of updating is greatest at $G_{i,b} = 0.5$, becoming smaller as $G_{i,b}$ moves away from 0.5, and hitting zero when the node freezes (for $G_{i,b}$ outside $[\epsilon, 1-\epsilon]$). As node i generates k units of traffic with probability P_{u_i} we could say that node i generates kP_{u_i} units of traffic on every iteration, making it clear that traffic reduction increases with $|G_{i,b} - 0.5|$.

In subsection 5.1.1 we looked into the distribution of $G_{i,1}$ over deterministic classic RBN's and noticed that for $k \geq 4$ the distribution can be estimated by a normal curve f_G, with mean $\mu = 0.5$ and standard deviation $\sigma_G = 2^{-(k/2+1)}$. We derived a standard deviation which took into account the variance when estimating $G_{i,1}$ with a finite memory, allowing us to compare theory with simulation results. Here we will only look at the exact $G_{i,1}$ distribution using σ_G (i.e. $G_{i,1}$ estimated with infinite memory). The distribution f_G can be used to find the initial traffic reduction on the network, hence give an upper bound. (We saw when observing the frozen component (subsection 5.1.2) that freezing of nodes

can spread throughout the network, thus reducing traffic further.) Let $G_{i,1} = G$ in the expressions f_G and P_{u_i} $(= P_u)$, for clearer notation.

$$\begin{aligned} f_G &= \frac{1}{\sigma\sqrt{2\pi}} \exp\left(\frac{-(G-\mu)^2}{2\sigma^2}\right) \\ &= \sqrt{\frac{2^{k+1}}{\pi}} \exp(-2^{k+1}(G-0.5)^2) \end{aligned}$$

Using f_G and P_u we can find the expectation of an arbitrary node updating, which is the same as the expected normalised traffic volume of the whole network (denoted ν_u) because,

$$\begin{aligned} \nu_u &= \frac{1}{Nk}\sum_{i=0}^{N-1} (\text{expectation of an arbitrary node updating}) \; k \\ &= \text{expectation of an arbitrary node updating} \end{aligned}$$

So,

$$\begin{aligned} \nu_u &= \int_0^1 f_G P_u dG \\ &= \sqrt{\frac{2^{k+1}}{\pi}} \frac{1}{1-2\epsilon} \left(\int_\epsilon^{1-\epsilon} \frac{1}{\exp(2^{k+1}(G-0.5)^2)} dG \right. \\ &\qquad \left. -\epsilon(1-\epsilon) \int_\epsilon^{1-\epsilon} \frac{1}{G(1-G)\,\exp(2^{k+1}(G-0.5)^2)} dG \right) \\ &= \sqrt{\frac{2^{k+1}}{\pi}} \frac{1}{1-2\epsilon} (F_1 - \epsilon(1-\epsilon)F_2) \end{aligned}$$

The integrals F_1 and F_2 can not be found in closed form and so we estimate F_1, F_2 numerically (using the trapezium rule). Figure 5.20 (left) shows the approximations of ν_u for $k = 4, 5, 9$ and $\epsilon = 0, 0.05, 0.1, \ldots, 0.5$. As k increases, ν_u approaches $1 - 2\epsilon$. This is predicted by the normal curves in figure 5.4: as k increases more nodes have $G_{i,1} \approx 0.5$, (the more nodes have $G_{i,1} \approx 0.5$, the more have $\tau_{i,0} = \tau_{i,1} = 1 - 2\epsilon$ and $P_{u_i} = 1 - 2\epsilon$, thus $\nu_u \to 1 - 2\epsilon$). We see a decrease in the gradient of ν_u as ϵ increases, this corresponds to nodes being frozen as the activity interval ($G_{i,1} \in [\epsilon, 1-\epsilon]$) contracts.

As mentioned earlier, ν_u is only an upper bound on the traffic reduction, so we took a snap shot of the traffic on simulated networks to get an idea how the normalised traffic volume (ν) developed with time, (figure 5.20 (right)). The numerical results were generated using three networks with the same in-degree for $k = 2, 3, 4, 5, 9$, $N = 100$. The networks were in a cycle when the accuracy algorithm was switched on ($\epsilon = 0, 0.05, 0.1, \ldots, 0.5$), they were then left to run

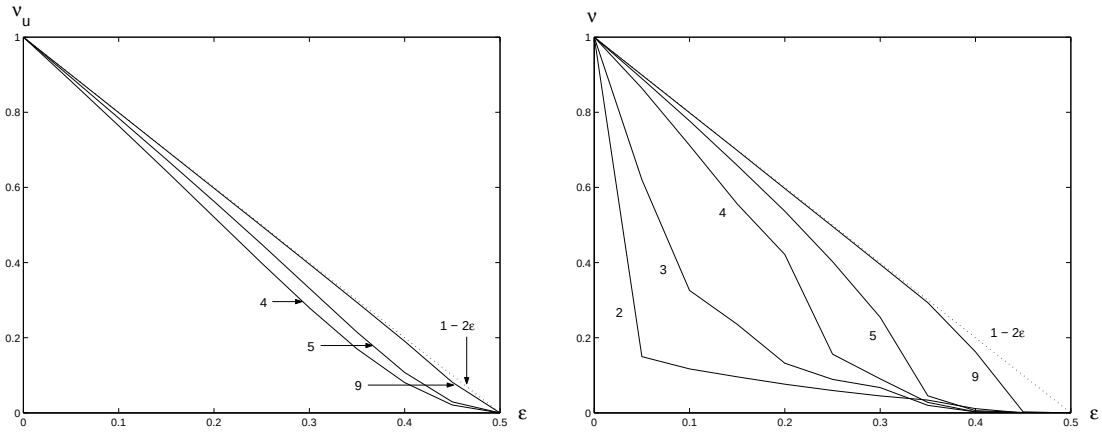

Figure 5.20: Left: approximations of ν_u for $k = 4, 5, 9$. Right: traffic snapshots of simulated networks $k = 2, 3, 4, 5, 9$ and $\epsilon = 0, 0.05, 0.1, \ldots, 0.5$.

for 20000 iterations before a traffic snapshot of 6000 iterations as taken. The traffic snapshot consists of finding the normalised traffic volume on each iteration (counting the number of requests/replies and dividing by Nk), then calculating the average of these over the 6000 iterations. This was then averaged with the other two traffic snapshots generated by the networks with the same k.

ν for $k = 4, 5, 9$ takes on a similar shape to their upper bound ν_u, they follow $\nu = 1 - 2\epsilon$ (not as closely as ν_u) then drop away from this path at a similar place to ν_u, though the drop is more dramatic. The curves for $k = 2, 3$ are similar, they have a sharp decrease followed by a more gradual decline. As the $k = 2$ accuracy networks freeze more often than not the optimal accuracy networks may be considered to be $k = 3$ networks; by optimal we mean that they often remain functional and benefit from a large traffic loss whilst retaining a high accuracy.

Chapter 6

Further work

6.1 Probabilistic RBN dynamics

In subsection 5.1.1 we looked at the distribution of $G_{i,1}$ (node output distributions) for deterministic classic RBNs, and generated a theoretical result to predict the distribution of $G_{i,1}$ as a function of in degree (for in-degree ≥ 4). The same could be done for semi-synchronous probabilistic networks (networks of nodes updating with probability m/N, $m \leq N$). Of course in order to give this a sensible meaning we would need to establish the stationarity of the state distribution in the network. The existence of loose attractors (subsection 1.2) in probabilistic RBNs support the possibility of stationarity, plus we can consider the whole RBN, with vector memories etc. included, as a Markov chain.

As the trajectories of probabilistic RBNs are closely related to those of their deterministic equivalents (subsection 4.2.2) one may expect the distribution of $H_{i,1}$ to be related to the distribution of $G_{i,1}$.

6.2 Correlation of input and output strings of Boolean functions

In section 4.1 we presented results of a numerical experiment supporting the idea that the output of a Boolean function with correlated inputs, becomes less correlated as the number of inputs increased. The Boolean function in the experiment was XOR (Boolean addition). Obviously looking at one function only gives a partial view of what is happening: for example, if the Boolean function returned 1 regardless of input then the correlation of the output would be 1 independent of the number of inputs and their statistical properties.

It would be interesting to explore the possibility of deriving a general expression for output correlation given input correlations (including higher order correlations and cross correlation, if necessary) and the Boolean function. (If such an expression were attainable, it could be used to improve the accuracy heuristic.)

6.3 Accuracy heuristic without freezing

A 'real' network is unlikely to be fixed, either in its topology or nodal behaviour (Boolean functions), so freezing nodes may be an impractical feature of the accuracy heuristic. This is easily fixed by setting the minimum re-evaluation probability greater than zero.

One could investigate the effects of this modification on the dynamics of accuracy networks, which must now not only depend on the inaccuracy (ϵ) but also on the minimum re-evaluation probability. When considering the frozen component (subsection 5.1.2) as a measure of dynamics, we might expect the the frozen component of the accuracy network (with non-freezing heuristic) to behave more like that of the semi-synchronous probabilistic network, than the heuristic with freezing.

6.4 Using the Accuracy heuristic to approximate Boolean functions

A boolean function can be represented as a tree: the leaf nodes supply the variables (or inputs) of the function, and the state of the root node is the value of the evaluated function (we looked at trees in section 4.1). Each node of the tree has one Boolean function; AND ($\wedge$), OR ($\vee$) or NOT (all Boolean functions can be re-written in terms of AND, OR, NOT: one method of doing this is the Quine McCluskey algorithm [25]). An example of the tree for the Boolean function $f = [(x_5 \wedge x_6) \wedge (x_7 \wedge x_8)] \wedge x_2$ is shown in figure 6.1.

Now consider letting the leaf nodes of a Boolean function tree output 1 with probability 1/2 (i.e. all (2^5, as the function has 5 variables) possible inputs have an equal probability of being realized). If we then impose the accuracy heuristic (section 3.1) on the tree and disregard nodes which are predecessors to frozen nodes (making the frozen node a leaf), we get a new tree describing a new Boolean function. What can we say about the new Boolean function as an approximation to the old one? i.e. how reliable an approximation is this new function and how

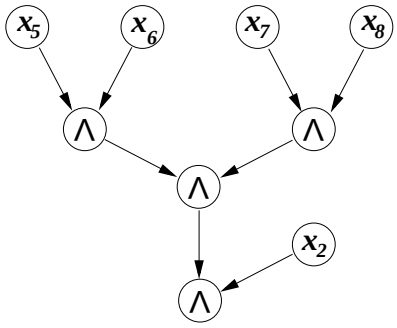

Figure 6.1: The tree representing the Boolean function $f = [(x_5 \wedge x_6) \wedge (x_7 \wedge x_8)] \wedge x_2$.

is this reliability related to ϵ (the inaccuracy the network)?

Returning to the example function, f: the left tree in figure 6.2 shows the probability of each node outputting 1 if all nodes in the tree are updated synchronously on every iteration. Now impose the heuristic on the tree (with $\epsilon = 0.1$), this alters the probability that nodes with predecessors are in state 1 (see subsection 3.2.1). The right tree in figure 6.2 gives the new probabilities. We see that the root node is frozen, suggesting f can be approximated by a constant, 0. As $f = 1$ with probability 1/32 the output of the new function will be inaccurate with probability 1/32.

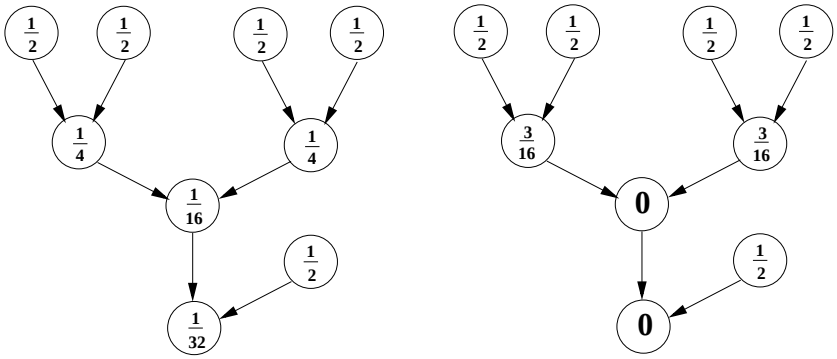

Figure 6.2: Probability of nodes having state 1 in the tree of f; deterministic network probabilities on the left, accuracy network probabilities on the right.

Now consider $g = [(x_5 \wedge x_6) \wedge (x_7 \wedge x_8)] \vee x_2$, this has an identical tree to f bar the root node, which is now OR rather than AND. The left tree in figure 6.3 shows the probabilities of nodes outputting 1 when the network is updated deterministically. The center tree gives the probabilities for the accuracy network. The right tree is the probabilities of nodes in the new function (g' which

approximates g) outputting 1. This suggests that $g' = 0 \vee x_2 \approx g$, and indeed we can see that $g' \neq g$ with probability 1/32.

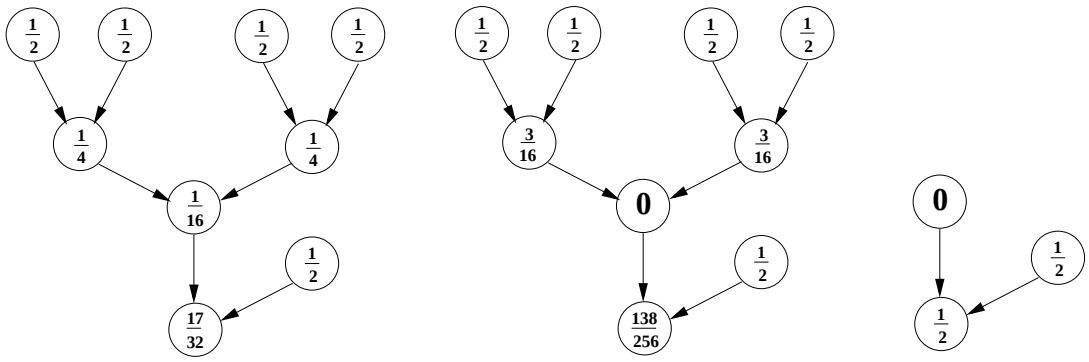

Figure 6.3: The probability of nodes in the tree of $g = [(x_5 \wedge x_6) \wedge (x_7 \wedge x_8)] \vee x_2$ outputting 1. Left: deterministic updating. Center: updating using the accuracy heuristic. Right: deterministic updating of $g' = 0 \vee x_2$.

It would be desirable to investigate this idea further, draw some general conclusions about the accuracy of the approximations, and automate the approximation process.

Appendix

Accuracy heuristic simulations

We have simulated accuracy networks with various topologies (loop, tree and random) recording the accuracy, traffic, frozen components and calculating the autocorrelations. Here we will discuss the basic accuracy heuristic simulation program, then talk a little about the modifications to the basic program which allowed us to obtain the numerical results.

The accuracy network simulation programs were written in Python which is a object orientated language based on C [21]. It was chosen because of its simple syntax, however Python is slow when dealing with large calculations.

Basic accuracy simulation

The network architecture and Boolean functions were set in a separate program from the main network simulation, and stored in a file for the simulation program to read. The initial condition (or state) was also generated in a separate program for the simulation to read, enabling us to carry out numerous tests on the same network with the same initial conditions.

The simulation program was only concerned with the evolution of states, the evolution of re-evaluation probabilities, and recording network properties (accuracy, traffic, frozen component and autocorrelation). For all the simulations discussed in this thesis the accuracy ($1 - \epsilon$, see section 3.1) is the same for all nodes. If simulations are to be run for several values of ϵ these are set in the simulation program and stored in a list. We will explain the use of the ϵ vector a little further on.

The initial re-evaluation probabilities ($\tau_{i,b}$ $b \in \{0, 1\}$) were set to 1 for each new simulation and fixed there for a transient period. During the transient period the simulation runs a deterministic network by evaluating all Boolean functions synchronously on every iteration. After this period the deterministic network

should be exhibiting long term behaviour (i.e. the network state should be on a cycle) which is useful when comparing network properties for various ϵ.

In order to run the accuracy heuristic each node needs to estimate $G_{i,1}$ (see section 3.1), thus the network is run deterministically for a further period determined by the size of the vector estimator or the weight of the scalar estimator (see section 3.2.2). Once nodes have $G_{i,1}$ estimates a copy of the current network state and all $G_{i,1}$ values are stored, this is considered to be the end of the deterministic phase.

To start the accuracy (probabilistic) phase the simulation uses the first value of ϵ (from the ϵ vector) and calculates the initial non-deterministic re-evaluation probabilities using $G_{i,1}$. Let the state of the network at time n be $Y^n = y_0^n y_1^n \dots y_{N-1}^n$. Each node updates its state using its Boolean function like so,
$y_i^{n+1} = f_i(y_{i,0}^n y_{i,1}^n \dots y_{i,k-1}^n) = f_i(Y_i^n)$ (recall there are k inputs per node). The loop below is performed for each iteration of the probabilistic network. This loop shows the $G_{i,1}$ estimation method which estimates $G_{i,1}$ directly (see subsection 3.2.1) as we use this most in the thesis.

for $i = 0, 1, \dots, N-1$ **:**

- R = random number in the unit interval

if $R < \tau_{i,y_i^n}$ **:**

- $y_i^{n+1} = f_i(Y_i^n)$
- update estimation of $G_{i,1}$ (and hence $G_{i,0}$)
- $\tau_{i,y_i^{n+1}}^{n+1} = 1 - \epsilon/(1 - G_{i,y_i^{n+1}})$ (equation (3.7))

else :

- $y_i^{n+1} = y_i^n$

At the end of the accuracy period the simulation checks for another value of ϵ (from the ϵ vector), if there is one it calculates another set of initial re-evaluation probabilities using the stored $G_{i,1}$ (from the end of the deterministic phase) and the new ϵ. The accuracy phase is then re-started using the state stored at the end of the deterministic phase as the initial condition.

For the tree topology some nodes have no predecessors (leaf nodes, see section **??**), the states of these nodes are chosen at random on every iteration, with probability P_i of being 1. The strings $(\dots, y_i^{n-1}, y_i^n, y_i^{n+1}, \dots)$ generated by leaf nodes

are iid. All other nodes in the tree have a Boolean function determining their state. In our accuracy tree simulations we chose to let the leaf nodes update on every iteration even after the accuracy heuristic had been enforced.

Recording the accuracy

To monitor the accuracy of a network we modified the basic loop (above) to include an accuracy counter $A = [a_0, a_1, \ldots, a_{N-1}]$, a_i is the accuracy of node i (A is a 'list'). Initially $a_i = 1$ as the network is run deterministically. Once the network updates probabilistically the value of a_i is altered using a running average (see the accuracy loop shown below). If a node updates it must be accurate and so we can simply increase a_i in this case. If the node does not update then it can only be accurate if $y_i^n = f_i(Y_i^n)$ is true, so this is tested and if it holds a_i is increased, otherwise a_i is decreased. Values of a_i were recorded in files for plotting. The accuracy loop is shown below,

for $i = 0, 1, \ldots, N-1$ **:**

- R = random number in the unit interval

if $R < \tau_{i,y_i^n}$ **:**

- $y_i^{n+1} = f_i(Y_i^n)$
- update estimation of $G_{i,1}$
- $\tau_{i,y_i^{n+1}}^{n+1} = 1 - \epsilon/(1 - G_{i,y_i^{n+1}})$
- $a_i = a_i(n-1)/n + 1/n$

else :

- $y_i^{n+1} = y_i^n$

if $y_i^n = f_i(Y_i^n)$ **:**

- $a_i = a_i(n-1)/n + 1/n$

else :

- $a_i = a_i(n-1)/n$

When calculating the average accuracy of a network, $\hat{a}_i$, only the accuracy of active nodes was used in the calculation. To do this we stored a_i as a 'dictionary', which is a Python object where each element is labelled enabling one to locate a node's accuracy in the absence of an ordered list. Once a node became inactive ($\tau_{i,b} = 0$) the accuracy of that node was struck from the dictionary, so the elements of dictionary A were accuracies of active nodes. At the end of each

iteration loop the average accuracy was calculated,

$$\hat{a}_i = \frac{1}{|A|} \sum_{a_i \in A} a_i$$

and written to a file for plotting.

Recording traffic volume

The traffic volume, ν, of an accuracy network was also calculated by a modification to the basic loop. For the deterministic network $\nu = 1$ as all nodes update on each iteration. In the accuracy phase ν was set to zero at the start of each iteration, then increasing by $1/N$ each time a node updated. The value of ν was written to a file for plotting at the end of each iteration. The traffic volume loop is shown below,

- $\nu = 0$

for $i = 0, 1, \ldots, N-1$ **:**

- $R =$ random number in the unit interval

if $R < \tau_{i,y_i^n}$ **:**

- $y_i^{n+1} = f_i(Y_i^n)$
- update estimation of $G_{i,1}$
- $\tau_{i,y_i^{n+1}}^{n+1} = 1 - \epsilon/(1 - G_{i,y_i^{n+1}})$
- $\nu = \nu + 1/N$

else :

- $y_i^{n+1} = y_i^n$

Recording the frozen component

The frozen component (subsection 5.1.2) was calculated outside the basic loop, by counting how long a node had been in a particular state (when the state of a node changed its counter was set to zero). If the count reached a specified minimum period the node was added to the frozen component but it could be removed if its state were to change whilst it was in there. The size of the frozen component was written to a file for plotting.

Autocorrelation calculations

For the autocorrelation calculations strings of node states
$(\ldots, y_i^{n-1}, y_i^n, y_i^{n+1} \ldots)$ were stored for analysis. Because if the length of these

strings (50000 elements for our calculations) a number of nodes were chosen at random for observation at the stare of simulations. Once the accuracy phase for a particular value of ϵ was completed the autocorrelation was calculated using equation (4.1), and written to a file for plotting.

Bibliography

[1] Aldana, M. Cluzel, P. (2003). A natural class of rhobust networks. *PNAS. 100.* `www.pnas.org/cgi/doi/10.1073/pnas.1536783100`.

[2] Bilke, S. Sjunnesson, F. (2001) Stability of the Kauffman model. *Phy Rev E 65*.

[3] Brzeźniak, Z. Zastawniak, T. Basic stochastic processes, *Springer undergraduate maths series*. London.

[4] Coulouris, G. Dollimore, J. Kindberg, T. (2001). Distributed Systems, concepts and design. *Pearson Education Ltd*. USA.

[5] Davis, P. (1994). Circulant Matrices. *Chelsea Pub Co*. New York.

[6] Derrida, B. Pomeau, Y. (1986). Random Networks of Automata: A Simple Annealed Approximation. *Europhys. Lett. 1 (2)*. 45-49.

[7] Flyvbjerg, H. (1988). An Order Perameter for Networks of Autometer. *Journal of Physics A. 21.* 955-960.

[8] Gershenson, C. (2003). Phase transitions in random Boolean networks with different updating schemes. *arXiv:nlin.AO/0311008 v1*.

[9] Harvey, I. Bossomier, T. (1997) Time Out of Joint: Attractors in Asynchronous Random Boolean Networks. `citeseer.nj.nec.com/47943.html`.

[10] Haykin, S. (1999). Neural Networks, a comprehensive foundation. *Prentice-Hall Inc.*. USA.

[11] Kauffamn, S. (1969). Metabolic Stabaility and Epigenesis in Randomly Constructed Genetic Nets. *J. Theoret. Biol. 22*. 437-467.

[12] Kauffamn, S. (1971). Gene regulation networks: A theory for their global structure and behavors. *Current topics in development biology. 6*. 145.

[13] Kauffman, S. (1974). The large scale structure and dynamics of gene control. *Journal of theoretacal biology. 44.* 167-190.

[14] Kauffman, S. (1993). Origins of order. *Oxford University Press* . NY.

[15] Luque, B. Solé, R. (2000). Lyapunov Exponents in Random Boolean Networks. *Physica A* . **284**. 33-45.

[16] Luque, B. Ballesteros, F. Muro, E. (2001) Self-organized critical random Boolean networks. *Phy Rev E 63*.

[17] Mesot, B. Teuscher, C. (2003). Critical values in asynchronous random Boolean networks. *Advances in artificial life, 7th Euroean conference. Springer - Verlag.* 367-376.

[18] Mockapetris, P. (2003). DNS growth has just begun: the challenge is to find new ways to use the almost limitless resource - Guest Column - domain name system. http://articles.findarticles.com/p/articles/mi_m0CMN/is_10_40/ai_109082402 .

[19] Di Paolo, E. (2001). Rhythmic and non-rhythmic attractors in asynchronous random Boolean network. *BioSystems 59*. 185-195.

[20] Rohlfshagen, P. Di Paolo, E. (2003). The circular toploogy of rhythem in asynchronous random Boolean networks. *BioSystems 73*. 141-152.

[21] Rossum, G. Python 2.4.1. *Python*. http://www.python.org/.

[22] Sawhill, B. Kauffman, S. (1997). Phase Transitions in Logic Networks. *Working papers, Santa Fe Institute*. http://econpapers.hhs.se/paper/wopsafiwp/97-05-038.htm.

[23] Solé, R. Luque, B. (1995). Phase Transitions and Antichaos in Generalised Kauffman Networks. *Physics Letters A. 196*. 331-334.

[24] Walker, C. Ashby, W. (1966). On temporal characteristics of behaviour in certain complex systems. *Kybernetik. 3.* 100-108.

[25] Wegener, I. (1987). The complexity of Boolean functions, *John Wiley and Sons Ltd*. UK.

[26] Weisstein, E. Jensen's Inequality. *MathWorld*. http://mathworld.wolfram.com/JensensInequality.html.

[27] Wessels, D. Fomenkov, M. Browlee, N. Claffy, K. (2004). Measurements and Laboratory Simulations of the Upper DNS Hieraachy. www.caida.org/outreach/papers/2004/dnspam.

[28] http://planetmath.org/encyclopedia/MoebiusFunction.html

[29] http://planetmath.org/encyclopedia/MobiusInversionFormula.html