# A Computational Model Based on Random Boolean Networks

Elena Dubrova
Royal Institute of Technology
Electrum 229, 164 46 Kista
Sweden
dubrova@kth.se

Maxim Teslenko
Royal Institute of Technology
Electrum 229, 164 46 Kista
Sweden
maximt@imit.kth.se

Hannu Tenhunen
Royal Institute of Technology
Electrum 229, 164 46 Kista
Sweden
hannu@imit.kth.se

## ABSTRACT

For decades, the size of silicon CMOS transistors has decreased steadily while their performance has improved. As the devices approach their physical limits, the need for alternative materials, structures and computation schemes becomes evident. This paper considers a computation scheme based on an abstract model of gene regulatory networks called *Random Boolean Networks*. Our interest in Random Boolean Networks is due to their attractive fault-tolerant features. The parameters of a network can be tuned so that it exhibits a robust behavior in which minimal changes in network's connections, values of state variables, or associated functions, typically cause no variation in the network's dynamics. A computation scheme based on random networks also seems to be appealing for emerging technologies in which it is difficult to control the growth direction or precise alignment, e.g. carbon nanotubes.

## Keywords

Random Boolean Network, attractor, Boolean function, fault-tolerance, carbon nanotubes

## 1. INTRODUCTION

A living cell could be considered as a molecular digital computer that configures itself as part of the execution of its code. The core of a cell is the DNA. DNA represents the information for building the basic components of cells as well as encodes the entire process of assembling complex components. By understanding how cells direct the assembly of their molecules, we can find ways to build chips that can self-organize, evolve and adapt to a changing environment.

The *gene regulatory network* is one of the most important signaling networks in living cells [1]. It is composed of the interactions of proteins with the genome. The major discovery related to gene regulatory networks was made in 1961 by French biologists François Jacob and Jacques Monod [24]. They found that a small fraction of the thousands of genes in the DNA molecule acts as tiny "switches". By exposing a cell to a certain hormone, these switches can be turned "on" or "off". The activated genes send chemical signals to other genes which, in turn, get either activated or repressed. The signals propagate along the DNA molecule until the cell settles down into a stable pattern.

Jacob and Monod's discovery showed that DNA is not just a blueprint for the cell, but rather an automaton which allows for the creation of different types of cells. It answered the long open question of how one fertilized egg cell could differentiate itself into brain cells, lung cells, muscle cells, and other types of cells that form a newborn baby. Each kind of cells corresponds to a different pattern of activated genes in the automaton.

In 1969 Stuart Kauffman proposed using Random Boolean Networks (RBN) as an abstract model of gene regulatory networks [25]. Each gene is represented by a vertex in a directed graph. An edge from one vertex to another implies a causal link between the two genes. The "on" state of a vertex corresponds to the gene being expressed. Time is viewed as proceeding in discrete steps. At each step, the new state of a vertex $v$ is a Boolean function of the previous states of the vertices which are the predecessors of $v$. Kauffman has shown that it is possible to tune the parameters of an RBN so that its statistical features match the characteristics of living cells and organisms [25]. The number of cycles in the RBN's state space, called *attractors*, corresponds to the number of different cell types. Attractor's length corresponds to the cell cycle time. Sensitivity of attractors to different kinds of disturbances, modeled by changing network connections, values of state variables, or associated functions, reflects the stability of the cell to damages, mutations, or virus attacks.

Later RBN were applied to the problems of cell differentiation [23], immune response [27], evolution [10], and neural networks [3, 2]. They have also attracted the interest of physicists due to their analogy with the disordered systems studied in statistical mechanics, such as the mean field spin glass [17, 15, 16].

In this paper, we investigate how RBN can be used for computing logic functions. Our interest to RBNs is due, one one hand, to their attractive fault-tolerant features. It is known that parameters of an RBN can be tuned so that the network exhibits a robust behavior, in which minimal changes in network's connections, values of state variables, or associated functions, typically cause no variations in the network's dynamics.

On the other hand, random networks are an appealing


*BIONETICS'07* December 10-13, 2007, Budapest, Hungary.

mathematical model for emerging nano-scale technologies in which it is difficult to control the growth direction or achieve precise assembly, e.g. carbon nanotubes. It has been demonstrated that random arrays of carbon nanotubes are much easier to produce compared to the ones with a fixed structure [31]. Random arrays of carbon nanotubes can be deposited at room temperature onto polymeric and many other substrates, which makes them a promising new material for *macroelectronics* applications. Macroelectronics is an important emerging area of technology which aims providing inexpensive electronics on polymeric substrates. Such electronics can be "printed" onto large-area polymeric films by using fabrication techniques similar to text and imaging printing, rather than conventional semiconductor fabrication technology. Applications of macroelectronics include lightweight flexible displays, smart materials or clothing, biological and chemical sensors, tunable frequency-selective surfaces, etc. Conventional semiconductors are not suitable for such applications because they are too expensive and require a crystalline substrate. The effort to develop organic semiconductors has achieved only moderate success so far, mostly because of the low-quality electron transport of organic semiconductors. Random arrays of carbon nanotubes provide a high-quality electron transport and therefore can be a much better alternative.

The paper is organized as follows. Section 2 gives a definition of RBNs and summarizes their properties. Section 3 describes how we can use RBNs for computing logic functions and addresses fault-tolerance issues. Section 4 presents a new algorithm for computing attractors in RBNs. Section 6 concludes the paper and discusses open problems.

## 2. RANDOM BOOLEAN NETWORKS

In this section, we give a brief introduction to Random Boolean Networks. For a more detailed description, the reader is referred to [2].

### 2.1 Definition of RBN

A *Random Boolean Network* (RBN) is a synchronous Boolean automaton with $n$ vertices. Each vertex $v$ has $k$ predecessors, assigned independently and uniformly at random from the set of all vertices, and an associated Boolean function $f_v : \{0,1\}^k \to \{0,1\}$. Functions are selected so that they evaluate to values 0 and 1 with given probabilities $p$ and $1-p$, respectively. Time is viewed as proceeding in discrete steps. At each step, the next value of the state variable $x_v$ associated with the vertex $v$ is a function of the previous values of the state variables $x_{u_i}$ associated with the predecessors of $v$, $u_i$, $i \in \{1, 2, \ldots, k\}$:

$$x_v^+ = f_v(x_{u_1}, x_{u_2}, \ldots, x_{u_k}),$$

The state of an RBN is defined by the ordered set of values of the state variables associated with its vertices.

An example of an RBN with $n = 10$ and $k = 2$ is shown in Figure 1. We use "$\cdot$", "$+$" and "$'$" to denote the Boolean operations AND, OR and NOT, respectively.

### 2.2 Frozen and chaotic phases

The parameters $k$ and $p$ determine the dynamics of an RBN. If a vertex controls many other vertices, and the number of controlled vertices grows in time, the RBN is said to be in a *chaotic phase* [29]. Typically such a behavior occurs for large values of $k \sim n$. The next states of the RBN

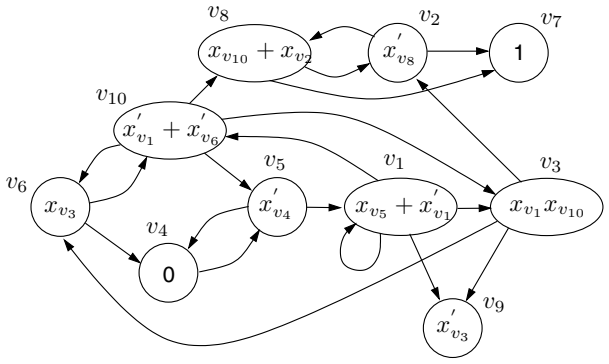


**Figure 1: Example of an RBN with $n = 10$ and $k = 2$. The next state of each vertex $v$ is given by $x_v^+ = f_v(x_{u_1}, x_{u_2})$, where $u_1$ and $u_2$ are the predecessors of $v$, and $f_v$ is the function associated to $v$.**

are random with respect to the previous ones. The dynamics of the network is very sensitive to changes in the values of state variables, associated Boolean function, or network connections.

If a vertex controls only a small number of other vertices and their number remains constant in time, the RBN is said to be in a *frozen phase* [21]. Usually, independently on the initial state, after a few steps, the network reaches a stable state. This behavior usually occurs for small values of $k$, such as $k = 0$ or 1.

There is a critical line between the frozen and the chaotic phases, when the number of vertices controlled by a vertex grows in time, but only up to a certain limit [4]. Statistical features of RBNs on the critical line are shown to match the characteristics of real cells and organisms [25, 26]. The minimal disturbances typically create no variations in the network's dynamics. Only some rare perturbations evoke radical changes.

For a given probability $p$, there is a critical number of inputs $k_c$ below which the network is in the frozen phase and above which the network is in the chaotic phase [17]:

$$k_c = \frac{1}{2p(1-p)}. \quad (1)$$

### 2.3 Attractors

An infinite sequence of consecutive states of a network is called a *trajectory*. A trajectory is uniquely defined by the initial state. Since the number of possible states is finite, all trajectories eventually converges to either a single state, or a cycle of states, called *attractor*. The *basin of attraction* of $A$ is the set of all trajectories leading to the attractor $A$. The *attractor length* is the number of states in the attractor's cycle.

It is possible to reduce the state space of an RBN by removing vertices belonging to its *stable core*. The stable core is defined by Flyvbjerg [20] as the set of vertices whose output value develops in time to a constant value that is independent of the initial state of the RBN. Bastola and Parisi [6] have observed that the state space can be further reduced by removing vertices which have no outputs. They introduced a notion on *relevant vertex*, which is a vertex

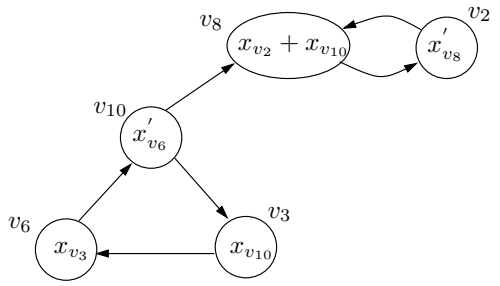


**Figure 2: Reduced network $G_R$ for the RBN in Figure 1.**

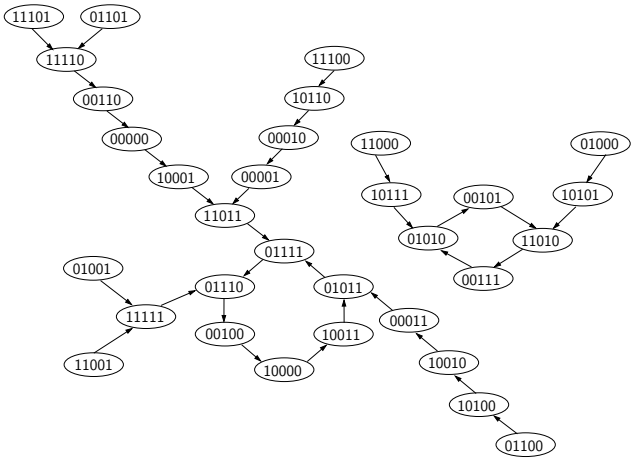


**Figure 3: State transition graph of the RBN in Figure 2. Each state is a 5-tuple $(x_{v_2}x_{v_3}x_{v_6}x_{v_8}x_{v_{10}})$.**

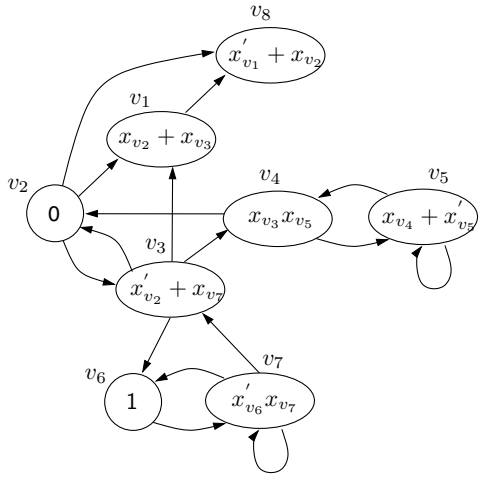


**Figure 4: Example of a network computing the 2-input AND.**

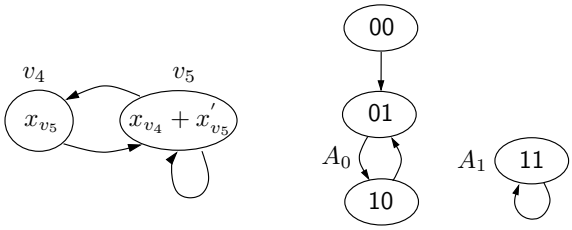


**Figure 5: (a) Reduced network for the RBN in Figure 4. (b) Its state transition graph. Each state is a pair $(x_{v_4}x_{v_5})$. There are two attractors: $A_1 = \{01, 10\}$ and $A_2 = \{11\}$.**

which has an influence on RBN's dynamics. Exact and approximate bounds on the size of the set of relevant vertices for different values of $k$ and $p$ have been given [20, 21, 4, 29, 6]. In the infinite size limit $n \to \infty$, in the frozen phase, the number of relevant vertices remains finite. In the chaotic phase, the number of relevant vertices is proportional to $n$. On the critical line, the number of relevant vertices scales as $n^{1/3}$ [32].

Let $G_R$ be the reduced network obtained from $G$ by removing non-relevant vertices. The reduced network for the example in Figure 1 is shown in Figure 2. Its state transition graph is given in Figure 3. Each vertex of the state transition graph represents a 5-tuple $(x_{v_2}x_{v_3}x_{v_6}x_{v_8}x_{v_{10}})$ of values of states on the relevant vertices $v_2$, $v_3$, $v_6$, $v_8$, $v_{10}$. There are two attractors: $\{01111, 01110, 00100, 10000, 10011, 01011\}$, of length six, and $\{00101, 11010, 00111, 01010\}$, of length four.

A number of algorithms for computing attractors in RBNs have been presented. Most of them are based on an explicit representation of the set of states on an RBN and therefore are applicable to networks with up to 32 relevant vertices only [32, 5, 36, 7]. The algorithm presented in [19] uses an implicit representation, namely Binary Decision Diagrams (BDDs) [11], and can handle larger RBNs. It is a very efficient algorithm which finds the set of states of all attractors simultaneously without computing the rest of the states. The algorithm presented in this paper is intended to complement the algorithm from [19] for the cases when the complete state space is necessary, e.g. if we want to compare dynamics of two RBNs.

For very larger RBNs, the median instead of the exact results are computed by simulation using the following technique [32]. Repeatedly, an initial state is chosen at random and the attractor reachable from this state is computed. If 1000 consecutive attempts yield no new attractor, the algorithm terminates. The resulting number is used as a lower bound on the number of attractors in the network.

In [18], it was shown that attractors of an RBN can be computed compositionally from the attractors of the connected components of the reduced network $G_R$. Two vertices are defined to be in the same component if and only if there is an undirected path between them.

# 3. COMPUTATIONAL SCHEME BASED ON RBNS

In this section we discuss how RBNs can be used for computing logic functions. One possibility is to use state variables of relevant vertices of a network to represent variables of the function, and to use attractors to represent the function's values.

To be more specific, suppose that we have an RBN $G$ with

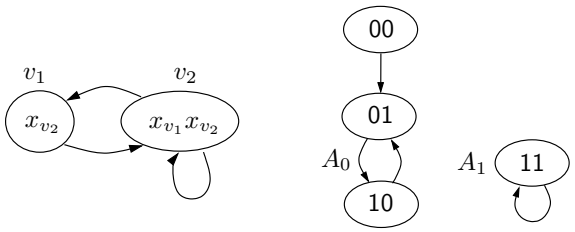


**Figure 6: An alternative reduced network for the 2-input AND.**

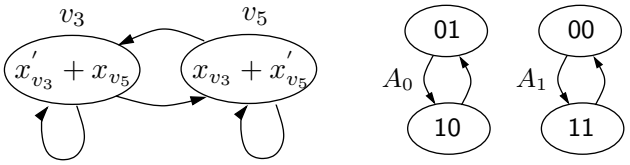


**Figure 7: (a) Reduced network for the RBN in Figure 4, after three mutations described in Section 3 have been applied. (b) Its state transition graph. Each state is a pair $(x_{v_3}x_{v_5})$. There are two attractors: $A_1 = \{01, 10\}$ and $A_2 = \{00, 11\}$.**

$r$ relevant vertices $v_1, \ldots, v_r$ and $m$ attractors $A_1, \ldots, A_m$. The basins of attractions of $A_i$'s partition the Boolean space $\{0,1\}^r$ into $m$ connected components via a dynamic process. Attractors constitute stable equilibrium points. We assign a value $i$, $i \in \{0, 1, \ldots, m-1\}$ to the attractor $A_i$ and assume that the set of points of the Boolean space corresponding to the states in the basin of attraction of $A_i$ is mapped to $i$. Then, $G$ defines the function $f : \{0,1\}^r \to \{0, 1, \ldots, m-1\}$ of variables $x_1, \ldots, x_r$, where the value of the variable $x_i$ corresponds to the state variable of the relevant vertex $v_i$. The mapping is unique up to the permutation of $m$ values of $f$. If $m = 2$, then $G$ represents a Boolean function.

The technique described above assumes that we have a way to access relevant vertices of an RBN and initialize them to given values (equivalent to assigning values to variables of a function), as well as that we have a way to recognize which logic value is assigned to an attractor (equivalent to reading output values of the function).

As an example, consider the RBN $G$ shown in Figure 4. The vertices $v_4$ and $v_5$ are relevant vertices, determining the dynamic of $G$ according to the reduced network in Figure 5(a). The state transition graph of the reduced network is shown in Figure 5(b). There are two attractors, $A_1$ and $A_2$. We assign the logic 0 to $A_1$ and the logic 1 to $A_2$. The initial states $00, 01$ and $10$ terminate in the attractor $A_1$ (logic 0) and the initial state 11 terminates in the attractor $A_2$ (logic 1). So, $G$ represents the 2-input Boolean AND.

As another example, consider the RBN in Figure 2 and its state transition graph in Figure 3. If we assign the logic 0 to the left-hand side attractor and the logic 1 to the right-had side one, then we get the following 5-variable Boolean function:

$$f(x_1, x_2, x_3, x_4, x_5) = x_2x'_3x'_5 + x'_2x_3x_4(x_1 + x_5).$$

A computation scheme based on RBNs inherits their attractive fault-tolerant features. Many experimental results confirm that RBNs are tolerant to faults, i.e. typically the number and length of attractors are not affected by small changes (see [2] for an overview). The following types of fault models are usually used:

- a predecessor of a vertex $v$ is changed, i.e. an edge $(u, v)$ is replaced by an edge $(w, v)$, $v, u, w \in V$;
- the value of a state variable is changed to the complemented value;
- Boolean function of a vertex is changed to a different Boolean function.

On one hand, the stability of RBNs is due to the large percentage of redundancy in the network. On the other hand, it is due to the non-uniqueness of the network representation. The same dynamic behavior can be achieved by many different RBNs. For instance, the 2-input AND gate can be implemented in many other ways than the one shown in Figure 4. For example, the reduced network in Figure 6 has the same state transition graph as the one in Figure 5.

Another interesting feature of RBN is their capability to evolve to a predefined target function. For example, suppose that the following three mutations are applied to the network in Figure 4:

1. edge $(v_4, v_5)$ is replaced by $(v_3, v_5)$;
2. edge $(v_2, v_3)$ is replaced by $(v_3, v_3)$;
3. edge $(v_7, v_3)$ is replaced by $(v_5, v_3)$.

After removing redundant vertices from the resulting modified network, we obtain the reduced network shown in Figure 7. Its state space has two attractors, $A_1$ and $A_2$. If we assign the logic 0 to $A_1$ and the logic 1 to $A_2$, then the initial states 00 and 11 terminate in 1, while 01 and 10 terminate in 0. So, the modified network implements the 2-input Boolean XNOR.

# 4. COMPUTATION OF ATTRACTORS

In order to realize a Boolean function using the method described above, attractors in the states space of an RBN have to be computed. In this section, we show that this can be done by using a technique similar to the fixed point computation during reachability analysis in model checking.

The main idea can be summarized as follows. Starting from an arbitrary state, forward reachability analysis is applied to find a state in some attractor $A$. Then, using this state as a final state, backward reachability analysis is performed to find the remaining states in the basin of attraction of $A$. The process in repeated starting from a state not previously visited until the complete state space is covered.

## 4.1 Transition relation

To be able to compute attractors in a large RBN, it is important to use an efficient representation for its set of states, and for the transition relation on this set. implementation, we use *Reduced Ordered Binary Decision Diagrams* (BDDs) [11].

A *transition relation* defines the next state values of the vertices in terms of the current state values. We derive

the transition relation in the standard way [13], by assigning every vertex $v_i$ of the network a state variable $x_{v_i}$ and making two copies of the set of state variables: $s = (x_{v_1}, x_{v_2}, \ldots, x_{v_r})$, denoting the variables of the current state, and $s^+ = (x^+_{v_1}, x^+_{v_2}, \ldots, x^+_{v_r})$, denoting the variables of the next state. Using this notation, the characteristic formula for the transition relation of an RBN is given by:

$$T(s, s^+) = \bigwedge_{i=1}^{r} (x^+_{v_i} \leftrightarrow f_i(x_{v_{i_1}}, x_{v_{i_2}})),$$

where $r$ is the number of relevant vertices, $f_i$ is the Boolean function associated with the vertex $v_i$ and $v_{i_1}$ and $v_{i_2}$ are the predecessors of $v_i$.

As an example, consider the reduced RBN in Figure 2 and its state transition graph in Figure 3. We have $s = (x_{v_1}, x_{v_2}, x_{v_5}, x_{v_7}, x_{v_9})$ and $s^+ = (x^+_{v_1}, x^+_{v_2}, x^+_{v_5}, x^+_{v_7}, x^+_{v_9})$. The transition relation is given by:

$$\begin{aligned} T(s, s^+) \quad = \quad & (x^+_{v_1} \leftrightarrow x'_{v_7}) \wedge (x^+_{v_2} \leftrightarrow x_{v_9}) \wedge (x^+_{v_5} \leftrightarrow x_{v_2}) \\ & \wedge (x^+_{v_7} \leftrightarrow (x_{v_1} + x_{v_9})) \wedge (x^+_{v_9} \leftrightarrow x'_{v_5}). \end{aligned}$$

## 4.2 Forward reachability

In traditional *forward reachability*, a sequence of formulas $F_i(s)$ representing the set of states that can be reached from a given set of initial states $Init$ in $i$ steps is computed as:

$$\begin{aligned} & F_0 = Init, \\ & F_{i+1}(s^+) = \exists s.(T(s, s^+) \wedge F_i(s)). \end{aligned}$$

The sequence generation is terminated when the fixed point is reached for some $p$:

$$\bigvee_{i=1}^{p} F_i(s) \rightarrow \bigvee_{i=1}^{p-1} F_i(s).$$

In an RBN, *any* state in the state space can be an initial state. We cannot start the reachability analysis from all states as the set of initial states, because then the fixed point is reached immediately. Instead, in our approach, forward reachability is started from a single state selected at random:

$$Init = \bigwedge_{i=1}^{r} (x_{v_i} \leftrightarrow init_i(x_{v_i})),$$

where $init_i(x_{v_i})$ is the initial value of the state variable $x_{v_i}$, $i \in \{1, \ldots, r\}$.

A sequence of consecutive states from $Init$ to an attractor can be quite long (up to $2^r$). Thus, it is not efficient to compute $F_i(s)$ for each value of $i$. To reduce the number of steps needed to reach an attractor, we use the *iterative squaring* technique [12]. Let $T^i(s, s^+)$ denote the transition relation describing the set of next states $s^+$ that can be reached from any current state $s$ in $i$ steps. For $i = 2$, $T^2(s, s^+)$ is computed as follows:

$$T^2(s, s^+) = \exists s^{++}.(T(s, s^{++}) \wedge T(s^{++}, s^+)). \qquad (2)$$

By applying squaring iteratively, we can obtain $T^{2^m}(s, s^+)$ in $m$ steps for any $m$.

One one hand, it cannot take more than $2^r$ steps to reach an attractor from any state. One the other hand, "overshooting" is not a problem because, once entered, an attractor is never left. Therefore, for any initial state $s$, the next state $s^+$ obtained by the transition defined by $T^{2^r}(s, s^+)$ is a state of an attractor.

We terminate the iterative computation of $T^{2^m}(s, s^+)$ if either $m$ becomes equal to $r$, or if

$$T^{2^m}(s, s^+) \;\rightarrow\; T^{2^{m-1}}(s, s^+),$$

for some $m \in \{1, \ldots, r-1\}$.

Using the resulting transition relation $T^{2^m}(s, s^+)$, $m \in \{1, \ldots, r\}$, we compute the set of states reachable from $Init$ in $2^m$ steps as:

$$F_{2^m}(s^+) = \exists s.(T^{2^m}(s, s^+) \wedge F_0(s)). \qquad (3)$$

Note that, with such an approach, we always reach a single state.

## 4.3 Backward reachability

The state given by $F_{2^m}(s^+)$ in equation (3) belongs to some attractor $A$. Next, we perform backward reachability to find the remaining states in the basin of attraction of $A$.

In traditional *backward reachability*, a sequence of formulas $B_i(s)$ representing the set of states from which a given set of final states $Final$ can be reached in $i$ steps is computed as:

$$\begin{aligned} & B_0 = Final, \\ & B_{i+1}(s) = \exists s^+.(T(s, s^+) \wedge B_i(s^+)). \end{aligned}$$

In our case, the set of final states consists of a single state $Final = F_{2^m}(s^+)$ (given by the equation (3)).

The sequence of consecutive states leading to $Final$ can be quite long (up to $2^r$). Thus, it is not efficient to compute $B_i(s)$ for each value of $i$. To reduce the number of steps needed to reach $Final$, we compute a transition relation $T_{0\ldots2^t}(s, s^+)$ which defines the set of all next states $s^+$ that can be reached from any current state $s$ in *up to* $2^t$ steps. This transition relation is used to obtain the the basin of attraction of $A$ by backward reachability. For $t \in \{0, \ldots, r\}$, $T_{0\ldots2^t}(s, s^+)$ is computed as follows:

$$\begin{aligned} & T_0(s, s^+) = \textstyle\bigwedge_{i=1}^{r} (x^+_{v_i} \leftrightarrow x_{v_i}), \\ & T_{0\ldots1}(s, s^+) = T(s, s^+) \vee T_0(s, s^+), \\ & T_{0\ldots2^t}(s, s^+) = T^2_{0\ldots2^{t-1}}(s, s^+), \end{aligned}$$

where $T^2_{0\ldots2^{t-1}}(s, s^+)$ is computed using the equation (2), and $T_0$ is the transition relation which assigns the next state of any state to be the state itself.

We terminate the iterative computation of $T_{0\ldots2^t}(s, s^+)$ if either $t$ becomes equal to $r$, or if

$$T_{0\ldots2^t}(s, s^+) \;\rightarrow\; T_{0\ldots2^{t-1}}(s, s^+),$$

for some $t \in \{1, \ldots, r-1\}$.

Using the resulting transition relation, we compute the set of states from which the state $Final$ is reachable in up to $2^t$ steps as

$$B_{0\ldots2^t}(s) = \exists s^+.(T_{0\ldots2^t}(s, s^+) \wedge B_0(s^+)),$$

where $B_0(s^+) = Final$. The resulting set $B_{0\ldots2^t}(s)$ is the basin of attraction of $A$.

The whole process is repeated starting from a state not belonging to any previously computed basin of attraction. The algorithm terminates when the complete state space is covered.

The pseudo-code of the algorithm described above is summarized in Figure 8.

**algorithm** FINDATTRACTORS ($T(s, s^+)$)
  $number\_of\_attractors = 0$;
  $C(s) = \emptyset$; /* set of states in computed basins of attraction */
  $T^1(s, s^+) = T(s, s^+)$;
  $m = 0$;
  **while** $m < r$ **do**
    $m$++;
    $T^{2^m}(s, s^+) = \exists s^{++}.(T^{2^{m-1}}(s, s^{++}) \wedge T^{2^{m-1}}(s^{++}, s^+))$;
    **if** $T^{2^m}(s, s^+) = T^{2^{m-1}}(s, s^+)$ **then**
      **break** ;
  **end while**
  $T_0(s, s^+) = \bigwedge_{i=1}^{r}(x^+_{v_i} \leftrightarrow x_{v_i})$;
  $T_{0\ldots1}(s, s^+) = T(s, s^+) \vee T_0(s, s^+)$;
  $t = 0$;
  **while** $t < r$ **do**
    $t$++;
    $T_{0\ldots2^t}(s, s^+) = T^2_{0\ldots2^{t-1}}(s, s^+)$;
    **if** $T_{0\ldots2^t}(s, s^+) = T_{0\ldots2^{t-1}}(s, s^+)$;
      **break** ;
  **end while**
  **while** $C(s) \neq U$ **do** /* $U$ is the complete state space */
    Pick up an initial state $F_0(s) \in C'(s)$;
    $F_{2^m}(s^+) = \exists s.(T^{2^m}(s, s^+) \wedge F_0(s))$;
    $B_{0\ldots2^t}(s) = \exists s^+.(T_{0\ldots2^t}(s, s^+) \wedge F_{2^m}(s^+))$;
    $C(s) = C(s) \cup B_{0\ldots2^t}(s)$;
    $number\_of\_attractors$++;
  **end while**
  **return** ($number\_of\_attractors$)
**end**

**Figure 8: Pseudocode of the presented algorithm for computing attractors in RBNs.**

Note that, when searching for a state in an attractor by forward reachability, $T_{0\ldots2^t}(s, s^+)$ can be used instead of $T_{2^m}(s, s^+)$. In this way, the calculation of $T_{2^m}(s, s^+)$ can be avoided without changing the overall procedure. The only difference is that, instead of a single state in an attractor, $F_{2^m}(s^+)$, a set of states is obtained.

As an example, consider the reduced RBN in Figure 2 and its state transition graph in Figure 3. We have $s = (x_{v_1}, x_{v_2}, x_{v_5}, x_{v_7}, x_{v_9})$ and $s^+ = (x^+_{v_1}, x^+_{v_2}, x^+_{v_5}, x^+_{v_7}, x^+_{v_9})$. The transition relation is given by:

$$\begin{aligned} T(s, s^+) &= (x^+_{v_1} \leftrightarrow x'_{v_7}) \wedge (x^+_{v_2} \leftrightarrow x_{v_9}) \wedge (x^+_{v_5} \leftrightarrow x_{v_2}) \\ &\quad \wedge (x^+_{v_7} \leftrightarrow (x_{v_1} + x_{v_9})) \wedge (x^+_{v_9} \leftrightarrow x'_{v_5}). \end{aligned}$$

After the first iteration of squaring, we get

$$\begin{aligned} T^2(s, s^+) &= \exists s^{++}.((x^{++}_{v_1} \leftrightarrow x'_{v_7}) \wedge (x^{++}_{v_2} \leftrightarrow x_{v_9}) \\ &\quad \wedge (x^{++}_{v_5} \leftrightarrow x_{v_2}) \wedge (x^{++}_{v_7} \leftrightarrow (x_{v_1} + x_{v_9})) \\ &\quad \wedge (x^{++}_{v_9} \leftrightarrow x'_{v_5}) \wedge (x^+_{v_1} \leftrightarrow (x^{++}_{v_7})') \\ &\quad \wedge (x^+_{v_2} \leftrightarrow x^{++}_{v_9}) \wedge (x^+_{v_5} \leftrightarrow x^{++}_{v_2}) \\ &\quad \wedge (x^+_{v_7} \leftrightarrow (x^{++}_{v_1} + x^{++}_{v_9})) \wedge (x^+_{v_9} \leftrightarrow (x^{++}_{v_5})'). \end{aligned}$$

Similarly, we compute $T^4(s, s^+)$, $T^8(s, s^+)$, and $T^{16}(s, s^+)$. None of $T^{2^m}(s, s^+)$ equals to $T^{2^{m-1}}(s, s^+)$ for $m \in \{1, \ldots, 4\}$. So, $T^{32}(s, s^+)$ is computed and the iterative squaring is terminated.

Suppose that the initial state is (11000), i.e.

$$\begin{aligned} Init &= (x_{v_1} \leftrightarrow 1) \wedge (x_{v_2} \leftrightarrow 1) \wedge (x_{v_5} \leftrightarrow 0) \wedge (x_{v_7} \leftrightarrow 0) \\ &\quad \wedge (x_{v_9} \leftrightarrow 0). \end{aligned}$$

Then, by substituting $T^{32}(s, s^+)$ and $F_0(s) = Init$ in (3) and simplifying the result, we get

$$\begin{aligned} F_{32}(s^+) &= (x_{v_1} \leftrightarrow 1) \wedge (x_{v_2} \leftrightarrow 1) \wedge (x_{v_5} \leftrightarrow 0) \\ &\quad \wedge (x_{v_7} \leftrightarrow 1) \wedge (x_{v_9} \leftrightarrow 0), \end{aligned}$$

i.e. the state (11010).

Backward reachability analysis gives us the remaining states in the basin of attraction, namely (11000), (10111), (01010), (00101), (00111), (10101), (01000). By repeating the process starting from, say (00001), we compute the second attractor. Since the complete state space is covered, the algorithm terminates.

| total number of vertices | average number of relevant vertices | average number of attractors |
|---|---|---|
| 10 | 5 | 2.69 |
| $10^2$ | 25 | 11.3 |
| $10^3$ | 93 | $24.1^*$ |
| $10^4$ | 270 | $89.7^*$ |
| $10^5$ | 690 | - |
| $10^6$ | 1614 | - |
| $10^7$ | 3502 | - |

**Table 1: Simulation results. Average values for 1000 RBNs with $k = 2$ and $p = 0.5$. "$*$" indicates that the average is computed only for successfully terminated cases.**

# 5. SIMULATION RESULTS

This section shows simulation results for RBNs of sizes from 10 to $10^7$ vertices with the parameters $k = 2$ and $p = 0.5$ (Table 1). Column 2 gives the average number of relevant vertices computed using the algorithm from [19]. Column 3 shows the average number of attractors in the resulting networks.

The simulation results show that, on random graphs, BDDs blow up more frequently than on sequential circuits. Currently, we cannot compute the exact number of attractors in most networks with $10^3$ vertices and larger. The number of attractors shown in column 5 for networks with $10^3$ and $10^4$ vertices is the average value computed for successfully terminated cases only. We did have occasional blow ups for networks with 100 vertices as well. The number of attractors shown in column 5 for networks with 100 vertices is the average value computed for 1000 successfully terminated cases.

# 6. CONCLUSION AND OPEN PROBLEMS

In this paper, we describe a computation scheme in which states of the relevant vertices of an RBN represent variables of the function, and attractors represent function's values. Such a computation scheme has attractive fault-tolerant features and it seems to be appealing for emerging nanotechnologies in which it is difficult to control the growth direction or achieve precise assembly.

We also present a new algorithm for computing attractors in RBN which uses Binary Decision Diagrams for representing the set of states of an RBN and the transition relation on this set. However, BDD representation runs out of memory for most networks with $10^3$ vertices and larger. We plan to

investigate possibilities for implementing the algorithm presented in [19] using Boolean circuits [9], rather than BDDs, and combined approaches [30, 35]. An Boolean circuit is a representation for Boolean formulas in which basic functions are restricted to AND, OR and NOT and all common subformulas are shared. Contrary to BDDs, Boolean circuits are not a canonical representation. Therefore, they normally take less memory to be stored, but need more time to be manipulated. We will also try reducing the state space of an RBN by detecting equivalent state variables [34] and by partitioning the transition relation [22].

In our future work, we will also investigate possibilities for enhancing RBNs as a model. RBNs have a number of drawbacks. First, input connectivity of gene regulatory networks is much higher than $k = 2$. For example, it is more than 20 in $\beta$-globine gene of humans and more than 60 for the platelet-derived growth factor $\beta$ receptor [2]. We will consider networks with a higher input connectivity $k$ and a smaller probability $p$, satisfying the equation (1)).

Second, using Boolean functions for describing the rules of regulatory interactions between the genes seems too simplistic. It is known that the level of gene expression depends on the presence of activating or repressing proteins. However, the *absence* of a protein can also influence the gene expression [2]. Using multiple-valued functions instead of Boolean ones for representing the rules of regulations could be a better option.

Third, the number of attractors in RBNs is a function of the number of vertices. However, organisms with a similar number of genes may have different numbers of cell types. For example, humans have 20.000-25.000 genes and more than 250 cell types [28]. The flower Arabidopis has a similar number of genes, 25.498, but only about 40 cell types [8]. We will investigate which other factors influence the number of attractors.

Another interesting problem is investigating networks with different types of connectivity. In RBNs, each vertex has an equal probability of being connected to other vertices. Alternatively, in cellular automata [14], each vertex is connected only to its immediate neighbors, and all connections are arranged in a regular lattice. Intermediate cases are possible, for example, in small-world networks [33] some connections are to distant vertices and some are to neighboring vertices.

# 7. REFERENCES


[1] B. Alberts, D. Bray, J. Lewis, M. Ra, K. Roberts, and J. D. Watson. *Molecular Biology of the Cell*. Garland Publishing, New York, 1994.

[2] M. Aldana, S. Coopersmith, and L. P. Kadanoff. Boolean dynamics with random couplings. http://arXiv.org/ abs/adap-org/9305001.

[3] H. Atlan, F. Fogelman-Soulie, J. Salomon, and G. Weisbuch. Random Boolean networks. *Cybernetics and System*, 12:103–121, 2001.

[4] U. Bastola and G. Parisi. The critical line of Kauffman networks. *J. Theor. Biol.*, 187:117, 1997.

[5] U. Bastola and G. Parisi. The modular structure of Kauffman networks. *Phys. D*, 115:219, 1998.

[6] U. Bastola and G. Parisi. Relevant elements, magnetization and dynamic properties in Kauffman networks: a numerical study. *Physica D*, 115:203, 1998.

[7] S. Bilke and F. Sjunnesson. Stability of the Kauffman model. *Physical Review E*, 65:016129, 2001.

[8] K. D. Birnbaum, D. E. Shasha, J. Y. Wang, J. W. Jung, G. M. Lambert, D. W. Galbraith, and P. N. Benfey. A global view of cellular identity in the Arabidopsis root. In *Proceedings of the International Conference on Arabidopsis Research*, Berlin, Germany, July 2004.

[9] P. Bjesse. DAG-aware circuit compression for formal verification. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 42–49, November 2004.

[10] S. Bornholdt and T. Rohlf. Topological evolution of dynamical networks: Global criticality from local dynamics. *Physical Review Letters*, 84:6114–6117, 2000.

[11] R. Bryant. Graph-based algorithms for Boolean function manipulation. *Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 35:677–691, August 1986.

[12] J. Burch, E. Clarke, D. E. Long, K. McMillan, and D. Dill. Symbolic Model Checking for sequential circuit verification. *Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 13(4):401–442, April 1994.

[13] J. Burch, E. Clarke, K. McMillan, D. Dill, and L. Hwang. Symbolic Model Checking: $10^{20}$ States and Beyond. In *Proceedings of the Fifth Annual IEEE Symposium on Logic in Computer Science*, pages 1–33, Washington, D.C., 1990. IEEE Computer Society Press.

[14] J. A. De Sales, M. L. Martins, and D. A. Stariolo. Cellular automata model for gene networks. *Physical Review E*, 55:3262–3270, 1997.

[15] B. Derrida and H. Flyvbjerg. Multivalley structure in Kauffman's model: Analogy with spin glass. *J. Phys. A: Math. Gen.*, 19:L1103, 1986.

[16] B. Derrida and H. Flyvbjerg. Distribution of local magnetizations in random networks of automata. *J. Phys. A: Math. Gen.*, 20:L1107, 1987.

[17] B. Derrida and Y. Pomeau. Random networks of automata: a simple annealed approximation. *Biophys. Lett.*, 1:45, 1986.

[18] E. Dubrova and M. Teslenko. Compositional properties of Random Boolean Networks. *Physical Review E*, 71:056116, May 2005.

[19] E. Dubrova, M. Teslenko, and A. Martinelli. Kauffman networks: Analysis and applications. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, November 2005.

[20] H. Flyvbjerg. An order parameter for networks of automata. *J. Phys. A: Math. Gen.*, 21:L955, 1988.

[21] H. Flyvbjerg and N. J. Kjaer. Exact solution of Kauffman model with connectivity one. *J. Phys. A: Math. Gen.*, 21:1695, 1988.

[22] D. Geist and I. Beer. Efficient model checking by automated ordering of transition relation partitions. In *Computer Aided Verification (CAV'94)*, pages 299–310, Stanford, July 1994. Springer-Verlag.

[23] S. Huang and D. E. Ingber. Shape-dependent control of cell growth, differentiation, and apoptosis: Switching between attractors in cell regulatory networks. *Experimental Cell Research*, 261:91–103, 2000.

[24] F. Jacob and J. Monod. Genetic regulatory mechanisms in the synthesis of proteins. *Journal of Molecular Biology*, 3:318–356, 1961.

[25] S. A. Kauffman. Metabolic stability and epigenesis in randomly constructed nets. *Journal of Theoretical Biology*, 22:437–467, 1969.

[26] S. A. Kauffman. *The Origins of Order: Self-Organization and Selection of Evolution*. Oxford University Press, Oxford, 1993.

[27] S. A. Kauffman and E. D. Weinberger. The nk model of rugged fitness landscapes and its application to maturation of the immune response. *Journal of Theoretical Biology*, 141:211–245, 1989.

[28] A. Y. Liu and L. D. True. Characterization of prostate cell types by cd cell surface molecules. *The American Journal of Pathology*, 160:37–43, 2002.

[29] B. Luque and R. V. Sole. Stable core and chaos control in Random boolean networks. *Journal of Physics A: Mathematical and General*, 31:1533–1537, 1998.

[30] S. M. Reddy, W. Kunz, and D. K. Pradhan. Novel verification framework combining structural and OBDD methods in a synthesis environment. In *Proceedings of the 32th ACM/IEEE Design Automation Conference*, pages 414–419, San Francisco, June 1995.

[31] E. S. Snow, J. P. Novak, P. M. Campbell, and D. Park. Random networks of carbon nanotubes as an electronic material. *Applied Physics Letters*, 81(13):2145–2147, 2003.

[32] J. E. S. Socolar and S. A. Kauffman. Scaling in ordered and critical random Boolean networks. http://arXiv.org/abs/cond-mat/0212306.

[33] S. H. Strogatz. Exploring complex networks. *Nature*, 410:268–276, 2001.

[34] C. A. J. van Eijk and J. A. G. Jess. Detection of equivalent state variables in finite state machine verification. In *1995 ACM/IEEE International Workshop on Logic Synthesis*, pages 3–35 – 3–44, Tahoe City, CA, May 1995.

[35] P. F. Williams, A. Biere, E. M. Clarke, and A. Gupta. Combining decision diagrams and SAT procedures for efficient symbolic model checking. In *Computer Aided Verification (CAV'00)*, pages 125–138, Chicago, IL, July 2000. Springer-Verlag.

[36] A. Wuensche. The DDlab manual, 2000. http://www.cogs.susx.ac.uk/ users/andywu/man_ contents.html.