Kauffman Networks: Analysis and Applications

Elena Dubrova Maxim Teslenko Andres Martinelli
Royal Institute of Technology, IMIT/KTH, 164 46 Kista, Sweden

***Abstract*— A Kauffman network is an abstract model of gene regulatory networks. Each gene is represented by a vertex. An edge from one vertex to another implies that the former gene regulates the latter. Statistical features of Kauffman networks match the characteristics of living cells. The number of cycles in the network's state space, called *attractors*, corresponds to the number of different cell types. The attractor's length corresponds to the cell cycle time. The sensitivity of attractors to different kinds of disturbances, modeled by changing a network connection, the state of a vertex, or the associated function, reflects the stability of the cell to damage, mutations and virus attacks. In order to evaluate attractors, their number and lengths have to be computed. This problem is the major open problem related to Kauffman networks. Available algorithms can only handle networks with less than a hundred vertices. The number of genes in a cell is often larger. In this paper, we present a set of efficient algorithms for computing attractors in large Kauffman networks. The resulting software package is hoped to be of assistance in understanding the principles of gene interactions and discovering a computing scheme operating on these principles.**

I. Introduction

The *gene regulatory network* is one of the most important signaling networks in living cells. It is composed of the interactions of proteins with the genome [1]. The major discovery related to gene regulatory networks was made in 1961 by French biologists François Jacob and Jacques Monod [2]. 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 Boolean networks for modeling gene regulatory networks [3]. 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 predecessors of v.

We discovered that many problems related to Kauffman networks are similar to the problems in logic synthesis and verification of electronic circuits. For example, the problem of finding relevant elements in Kauffman networks [4] is similar to the problem of removing redundancy in sequential logic circuits [5]. The problem of identifying state cycles in Kauffman networks [6] is related to the problem of image computation in model checking [7].

After examining the state-of-the-art in Kauffman networks, we found that existing methods for their analysis are quite immature compared to the approaches used in logic synthesis and verification. There are efficient techniques for removing redundancy from a circuit with millions of gates [5] and for verifying finite state machines with 10^{20} states [8]. The programs available for computing state cycles in Kauffman networks can only deal with networks with less than 32 relevant vertices [9], [10], [11], [12]. The number of genes in a cell is often larger. For example, the tiny worm Caenorhabditis elegans has 19.099 genes. A small flower in the mustard family, Arabidopis, has 25.498 genes [13].

To bridge this gap, we developed algorithms for redundancy removal and partitioning for Kauffman networks that have linear-time complexity and are feasible for networks with millions of vertices [14], [15], [16]. These algorithms are first steps towards solving the more central problem of computing state cycles in large Kauffman networks, which is addressed in this paper.

II. Kauffman Networks

In this section, we give a brief introduction to Kauffman networks. For a more detailed description, the reader is referred to [17].

A. Definition of Kauffman Networks

Kauffman networks are a class of *random nk-Boolean networks* [18]. A random nk-Boolean network is a synchronous Boolean automaton with n vertices. Each vertex has exactly k incoming edges, assigned at random, and an associated Boolean function. Functions are selected so that they evaluate to the values 0 and 1 with given probabilities p and $1-p$, respectively. 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 predecessors of v.

A Kauffman network is a random nk-Boolean network with $k = 2$ and $p = 0.5$, i.e. each vertex has two predecessors and Boolean functions are assigned to vertices independently and uniformly at random from the set of 16 possible 2-variable Boolean functions [19]. The state σ_{v_i} of a vertex v_i at time $t+1$ is determined by the states of its predecessors v_l and v_r, $i,l,r \in \{1,2,\ldots,n\}$, as:

$$\sigma_{v_i}(t+1) = f_{v_i}(\sigma_{v_l}(t), \sigma_{v_r}(t))$$

where $f_{v_i} : \{0,1\}^2 \to \{0,1\}$ is the Boolean function associated to v_i. The vector $(\sigma_{v_1}(t), \sigma_{v_2}(t), \ldots, \sigma_{v_n}(t))$ represents the state of the network at time t. An example of a Kauffman network with ten vertices is shown in Figure 1. We use "·", "+" and "′" to denote the Boolean operations AND, OR and NOT, respectively.

B. Frozen and chaotic phases

The parameters k and p determine the dynamics of the network. 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 [20]:

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

If a network is in the *frozen phase*, then, independently of the initial state, a stable state is reached after a few steps [21]. Small changes in network's connections, states of vertices, or associated Boolean functions, typically create no variations in the network's dynamics.

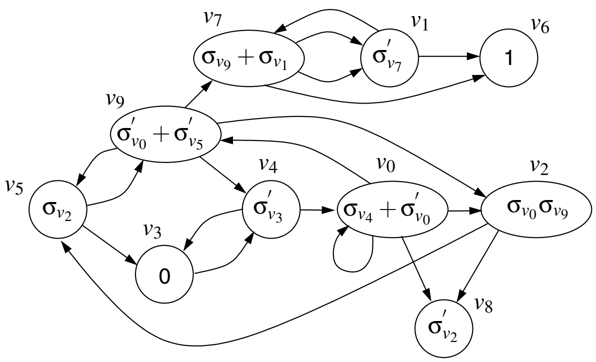

Fig. 1. Example of a Kauffman network. The state of a vertex v_i at time $t+1$ is given by $\sigma_{v_i}(t+1) = f_{v_i}(\sigma_{v_l}(t), \sigma_{v_r}(t))$, where v_l and v_r are the predecessors of v_i, and f_{v_i} is the Boolean function associated to v_i.

In the *chaotic phase*, the length of state cycles is of order of 2^n. The dynamics of the network is very sensitive to changes in network's connections, states of vertices, or associated Boolean functions [22].

On the critical line between the frozen and the chaotic phases, the network exhibits *self-organized critical behavior*, ensuring both stability and evolutionary improvements [23]. Statistical features of random *nk*-Boolean networks on the critical line are shown to match the characteristics of real cells and organisms [3], [24], [17]. For $p = 0.5$, the critical number of inputs is $k_c = 2$, so Kauffman networks are on the critical line.

Apart from gene regulatory networks, Kauffman networks have been applied to the problems of cell differentiation [25], immune response [26], and evolution [27]. They have also attracted the interest of physicists due to their analogy with disordered systems studied in statistical mechanics, such as the mean field spin glass [28].

C. Attractors

Since the number of possible states of a Kauffman network is finite (up to 2^n), any sequence of consecutive states of a network eventually converges to either a single state, or a cycle of states, called *attractor*. The number and length of attractors represent two important parameters of the cell modeled by a Kauffman network. The number of attractors corresponds to the number of different *cell types*. For example, humans have 20.000-25.000 genes (the exact number is not known yet) and about 250 cell types [29]. The attractor's length corresponds to the *cell cycle time*. Cell cycle time refers to the amount of time required for a cell to grow and divide into two daughter cells. The length of the total cell cycle varies for different types of cells.

The human body has a sophisticated system for maintaining normal cell repair and growth. The body interacts with cells through a feedback system that signals a cell to enter different phases of the cycle [30]. If a person is sick, e.g suffers from cancer, then this feedback system does not function normally and cancer cells enter the cell cycle independently of the body's signals. The number and length of attractors of a Kauffman network serve as indicators of the health of the cell modeled by the network [6]. The sensitivity of attractors to different kinds of disturbances, modeled by changing the state of a vertex, the associated Boolean function, or a network connection, reflects the stability of the cell to damage, mutations and virus attacks.

In order to evaluate attractors, their number and length have to be computed. This problem is the major problem in the analysis of Kauffman networks, for which no efficient solution is found so far. Available algorithms for exact computation of attractors can only handle networks with less than 32 non-redundant vertices [9], [10], [11], [12]. For larger networks, the median instead of the exact values on the number of attractors is computed using the following technique [12]. 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.

```
algorithm RemoveRedundant (V,E)
    /* I. Simplification of vertices with one predecessor */
    for each v ∈ V do
        if two incoming edges of v come from the same vertex then
            Simplify f_v;
    /* II. Constant propagation */
    R_1 = Ø;
    for each v ∈ V do
        if f_v is a constant then
            Append v at the end of R_1;
    for each v ∈ R_1 do
        for each u ∈ S_v − R_1 do
            Simplify f_u by substituting constant f_v;
            if f_u is a constant then
                Append u at the end of R_1;
    Remove all v ∈ R_1 and all edges connected to v;
    /* III. Simplification of vertices with 1-variable functions */
    for each v ∈ V do
        if f_v is a 1-variable function then
            Remove the edge (u,v), where u is the
            predecessor of v on which v does not depend;
    /* IV. Elimination of vertices with no outputs */
    R_2 = Ø;
    for each v ∈ V do
        if S_v = Ø then
            Append v at the end of R_2;
    for each v ∈ R_2 do
        for each u ∈ P_v − R_2 do
            if all ancestors of u are in R_2 then
                Append u at the end of R_2;
    Remove all v ∈ R_2 and all edges connected to v;
end
```

Fig. 2. The algorithm for finding redundant vertices in Kauffman networks.

III. Redundancy Removal

Redundancy is an essential feature of biological systems, ensuring their correct behavior in presence of internal or external disturbances. An overwhelming percentage (about 95%) of DNA of humans is redundant to the metabolic and developmental processes. Such "junk" DNA is believed to act as a protective buffer against genetic damage and harmful mutations, reducing the probability that any single, random offense to the nucleotide sequence will affect the organism [31].

In the context of Kauffman networks, redundancy is defined as follows. Let $G = (V, E)$ be a Kauffman network, where V is the set of vertices and $E \subseteq V \times V$ is the set of edges connecting the vertices.

Definition 1: A vertex $v \in V$ of a Kauffman network G is redundant if the network obtained from G by removing v has the same number and length of attractors as G.

If a vertex in not redundant, it is called *relevant* [9].

In [9], an algorithm for computing the set of all redundant vertices was presented. This algorithm has a high complexity, and therefore is only applicable to small Kauffman networks with up to a hundred vertices. In [15], we presented an algorithm RemoveRedundant (Figure 2), which quickly finds structural redundancy and some simple cases of functional redundancy. The phases II and IV of RemoveRedundant are similar to the *decimation procedure*

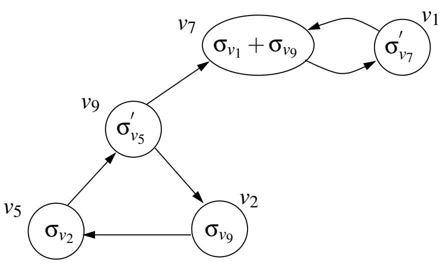

Fig. 3. Reduced network G_R for the Kauffman network in Figure 1.

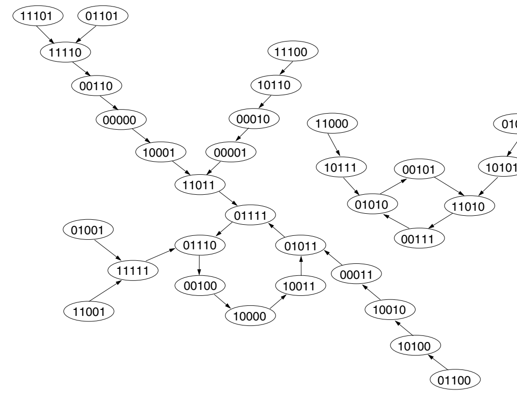

Fig. 4. State transition graph of the Kauffman network in Figure 3. Each state is a 5-tuple $(\sigma(v_1)\sigma(v_2)\sigma(v_5)\sigma(v_7)\sigma(v_9))$.

of [11], although a detailed comparison is hard to do because no pseudocode is shown in [11]. The ordering of the phases of the algorithm is very important. For example, if the phase IV is performed before the phase II, then usually less redundant vertices are found.

Let $P_v = \{u \in V \mid (u,v) \in E\}$ be a set of *predecessors* of $v \in V$ and $S_v = \{u \in V \mid (v,u) \in E\}$ be a set of *successors* of v.

RemoveRedundant first checks whether there are vertices v with two incoming edges coming from the same vertex. If yes, the associated functions f_v are simplified.

Then, RemoveRedundant classifies as redundant all vertices v whose associated function f_v is constant 0 or constant 1. Such vertices are collected in a list R_1. Then, for every vertex $v \in R_1$, successors of v are visited and the functions associated to the successors are simplified. The simplification is done by substituting the constant value of f_v in the function of the successor u. If as a result of the simplification the function f_u reduces to a constant, then u is appended to R_1.

Second, RemoveRedundant finds all vertices whose associated function f_v is a single-variable function. The edge between v and the predecessor of v which v does not depend on is removed.

Next, RemoveRedundant classifies as redundant all vertices which have no successors. Such vertices are collected in a list R_2. For every vertex $v \in R_2$, both predecessors of v are visited. If all successors of some predecessor $u \in P_v$ are redundant, u is appended at the end of R_2.

The worst-case time complexity of RemoveRedundant is $O(|V|+|E|)$, where $|V|$ is the number of vertices and $|E|$ is the number of edges in G.

As we mentioned before, RemoveRedundant might not identify all cases of functional redundancy. For example, a vertex may have a constant output value due to the correlation of its input variables. For example, if a vertex v with an associated OR (AND) function has predecessors v_l and v_r with functions $f_{v_l} = \sigma_{v_j}$ and $f_{v_r} = \sigma'_{v_j}$, then the value of f_v is always 1 (0). Such cases of redundancy are not detected by RemoveRedundant.

Let G_R be the reduced network obtained from G by removing redundant vertices. The reduced network for the example in Figure 1 is shown in Figure 3. Its state transition graph is given in Figure 4. Each vertex of the state transition graph represents a 5-tuple $(\sigma(v_1)\sigma(v_2)\sigma(v_5)\sigma(v_7)\sigma(v_9))$ of values of states on the relevant vertices v_1, v_2, v_5, v_7, v_9. There are two attractors: $\{01111, 01110, 00100, 10000, 10011, 01011\}$, of length six, and $\{00101, 11010, 00111, 01010\}$, of length four. By Definition 1, by removing redundant vertices we do not change the total number and length of attractors in a Kauffman network. Therefore, G_R has the same number and length of attractors as G.

IV. Partitioning

The vertices of G_R induce a number of connected components.

Definition 2: Two relevant vertices are in the same component if and only if there is an undirected path between them.

A path is called *undirected* if it ignores the direction of edges.

Connected components can be computed in $O(|V|+|E|)$ time, where $|V|$ is the number of vertices and $|E|$ is the number of edges of G_R, using the following algorithm [32]. To find a connected component number i, the function ComponentSearch(v) is called for a vertex v which has not been assigned to a component yet. ComponentSearch does nothing if v has been assigned to a component already. Otherwise, ComponentSearch assigns v to the component i and calls itself recursively for all predecessors and successors of v. The process repeats with the counter i incremented until all vertices are assigned.

In [16], we have shown that attractors of a Kauffman network can be computed compositionally from the attractors of the connected components of G_R. Let $\{G_1, G_2, \ldots, G_p\}$ be the set of components of G_R, N_i be the number of attractors of G_i, L_{ij} be the length of the jth attractor G_i and $I = I_1 \times I_2 \times \ldots \times I_p$ be the Cartesian product of sets $I_i = \{i_1, i_2, \ldots, i_{N_i}\}$, $i = \{1, 2, \ldots, p\}$, $j = \{1, 2, \ldots, N_i\}$. Then, the total number of attractors in G_R is given by

$$N = \sum_{\forall (i_1,\ldots,i_p) \in I} \prod_{j=2}^{p} (((L_{1i_1} \bullet L_{2i_2}) \bullet L_{3i_3}) \ldots \bullet L_{j-1 i_{j-1}}) \circ L_{j i_j}$$

where "$\bullet$" is the least common multiple operation and "$\circ$" is the greatest common divisor operation. The maximum length of attractors is given by

$$L_{max} = \max_{\forall (i_1,\ldots,i_p) \in I} ((L_{1i_1} \bullet L_{2i_2}) \bullet L_{3i_3}) \ldots \bullet L_{p i_p}$$

where "$\bullet$" is the least common multiple operation.

V. Computation of Attractors

To be able to compute attractors in a large Kauffman network, it is important to use an efficient representation for its set of states, and for the transition relation on this set. In our current implementation, we use *Reduced Ordered Binary Decision Diagrams* (ROBDDs) [33].

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 [8], 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 a Kauffman network 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 Kauffman network in Figure 3 and its state transition graph in Figure 4. 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:

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

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 exactly i steps. For $i = 2$, $T^2(s,s^+)$ is computed as follows:

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

By applying squaring iteratively, we can obtain $T^{2^i}(s,s^+)$ in i steps for any i [34].

On one hand, for any Kauffman network with r relevant vertices, 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.

Let $F_i(s)$ denote the set of states reachable from a given set of initial states in i steps. Using the transition relation $T^{2^r}(s,s^+)$, we can compute the set of states $F_{2^r}(s)$ that can be reached from *any* state in 2^r steps as:

$$F_{2^r}(s^+) = \exists s.T^{2^r}(s,s^+).$$

$F_{2^r}(s^+)$ represents the set of states of *all* attractors. It remains to distinguish between different attractors. This can be done by simulation as follows. An arbitrary state σ of $F_{2^r}(s^+)$ is picked up and the sequence of σ's next states is followed until σ is reached again. The sequence of visited states represents an attractor. This process is repeated starting from a state of $F_{2^r}(s^+)$ not visited yet until $F_{2^r}(s^+)$ is covered.

Our simulation results show that the number and lengths of attractors in a Kauffman network with n vertices are of order of $\sqrt{n}$. Therefore, the number of states in $F_{2^r}(s^+)$ is of order of $\sqrt{n} \cdot \sqrt{n} = n$. Thus, enumerating all states of $F_{2^r}(s^+)$ is feasible in practice.

VI. Simulation Results

This section shows simulation results for Kauffman networks of sizes from 10 to 10^7 vertices (Table I). Column 2 gives the average number of relevant vertices computed using RemoveRedundant. Column 3 shows the average size of the largest connected component of the subgraph G_R induced by the relevant vertices and column 4 gives the average number of components. Column 5 shows the average number of attractors.

The simulation results show that we need to find a better way of partitioning. Currently, the size of the largest component of the subgraph induced by the relevant vertices (column 3) is $\Theta(r)$, where r is the number of relevant vertices in the subgraph, i.e. we observe so called "giant" component phenomena [35]. A technique resulting in a more balanced partitioning is needed.

total number of vertices	average number of relevant vertices	average size of the largest component	average number of components	average number of attractors
10	5	5	1.1	2.67
10^2	25	25	1.4	11.7
10^3	93	92	1.8	23.9*
10^4	270	266	2.4	-
10^5	690	682	3.1	-
10^6	1614	1596	3.7	-
10^7	3502	3463	4.3	-

TABLE I

Simulation results. Average values for 1000 networks.

Another problem is that, on random graphs, ROBDDs 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 vertices (marked with "$*$") 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. In our future work, we plan to investigate possibilities for implementing the algorithm presented in Section V using Boolean circuits [36], [37], [38], [39], rather than ROBDDs, and combined approaches [40], [41]. We will also try reducing the state space by detecting equivalent state variables [42] and by partitioning the transition relation [43].

VII. Applications

In this section we present some ideas on how Kauffman networks can be used for implementing Boolean functions and for achieving fault-tolerance. The ideas we describe are preliminary, more research is needed to justify them.

A. Implementing logic functions by Kauffman networks

An interesting direction of research is investigating how Kauffman networks can be used for implementing logic functions. One possibility is to use the states of relevant vertices of a network to represent variables of the function, and to use the attractors to represent the function's values.

To be more specific, suppose that we have a Kauffman network G with r relevant vertices $v_1, \ldots, v_r$ and m attractors $A_1, A_2, \ldots, A_m$. The basins of attractions of A_i's partition the Boolean space B^r into m connected components. We assign a value i, $i \in \{0, 1, \ldots, m-1\}$ to the attractor A_i and assume that the set of minterms represented by the states in the basin of attraction of A_i is mapped to k. Then, G implements 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 of relevant vertex v_i. The mapping is unique up to permutation of m output values of f. If $m = 2$, then G implements a Boolean function.

As an example, consider the Kauffman network G shown in Figure 5. The vertices v_4 and v_5 are relevant vertices, determining the dynamic of G according to the reduced network in Figure 6(a). The state transition graph of the reduced network is shown in Figure 6(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 implements the 2-input Boolean AND.

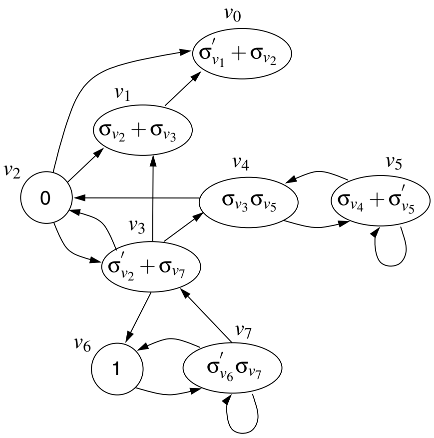

Fig. 5. Example of a network implementing the 2-input AND.

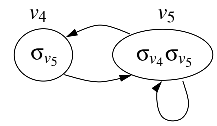

Fig. 7. An alternative reduced network for the 2-input AND.

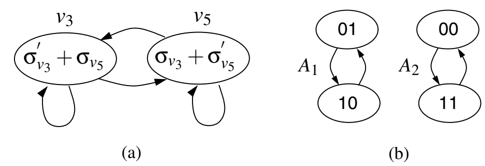

Fig. 8. (a) Reduced network for the Kauffman network in Figure 5, after three mutations described in Section VII-C have been applied. (b) Its state transition graph. Each state is a pair $(\sigma(v_3)\sigma(v_5))$. There are two attractors: $A_1 = \{01, 10\}$ and $A_2 = \{00, 11\}$.

B. Stability

Extensive experimental results confirm that Kauffman networks are tolerant to faults, i.e. typically the number and length of attractors are not affected by small changes [24], [17]. The following types of fault models are used to model the effects of diseases, mutations, or injuries on a cell:

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

On one hand, the stability of Kauffman networks is due to the large percentage of redundancy in the network. $\Theta(n-\sqrt{n})$ of n vertices are typically redundant. On the other hand, the stability is due to the non-uniqueness of the network representation. The same dynamic behavior can be achieved by many different Kauffman networks. For instance, the 2-input AND gate could be implemented in many other ways than the one shown in Figure 5. For example, the reduced network in Figure 7 has the same state transition graph as the one in Figure 6.

C. Evolvability

An essential feature of living organisms is their capability to adapt to a changing environment. Kauffman networks have been shown to be successful in evolving to a predefined target function.

As an example, suppose that the following three mutations are applied to the network in Figure 5:

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);

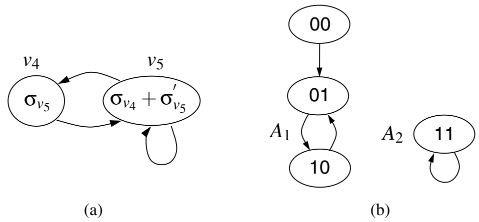

Fig. 6. (a) Reduced network for the Kauffman network in Figure 5. (b) Its state transition graph. Each state is a pair $(\sigma(v_4)\sigma(v_5))$. There are two attractors: $A_1 = \{01, 10\}$ and $A_2 = \{11\}$.

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 8. 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.

The example given above is intended to demonstrate that an evolution from one functionality to another is possible.

VIII. Conclusion and Future Work

This paper presents a set of algorithms for the analysis of Kauffman networks. Redundancy removal and partitioning algorithms have been presented previously in [14], [15], [16]. The algorithm for computing attractors is a new contribution, as well as the proposed applications.

We would like to stress that the major challenge is the *size* of the networks we are targeting. Small Kauffman networks are of theoretical interest only. They cannot adequately model gene interactions of living cells. We aim at developing a practical software package, applicable to real world size problems.

A software package that can model gene interactions is of primary importance to biology and medicine. Such a package will provide a framework for obtaining simulation results that can be independently evaluated by *in vivo* experiments. It can be used for various purposes, including:

1) to study the effects of diseases, mutations, or injuries on a cell;
2) to infer gene interactions that produce abnormal cells, e.g. cancer;
3) to understand the process of aging of a cell over time.

In the future, we will also investigate possibilities for enhancing Kauffman networks as a model. Kauffman networks 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 β-globine gene of humans and more than 60 for the platelet-derived growth factor β receptor [17]. 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 [17]. 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 Kauffman networks 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 [29]. The flower Arabidopis has a similar number of genes, 25.498, but only about 40 cell types [44]. We will investigate which other factors influence the number of attractors.

As a longer-term goal, we will attempt to develop a computing scheme based on the principles of gene interactions. A living cell is, essentially, a molecular computer that configures itself as part of the execution of its code. By understanding how genes interact with each other, we might find a way to build a novel type of computer chips. As silicon transistor technology approaches nano-meter dimensions and its speed and integration slow down, the need for new ways of computing becomes more and more evident.

References

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

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

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

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

[5] M. Berkelaar and K. M. van Eijk, "Efficient and effective redundancy removal for million-gate circuits," in *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition*, p. 1088, IEEE Computer Society Press, 2002.

[6] Z. Somogyvari and S. Payrits, "Length of state cycles of random boolean networks: an analytic study," *Journal of Physics A: Mathematical and General*, vol. 33, pp. 6699–6706, 2000.

[7] K. L. McMillan, *Symbolic Model Checking*. Boston, MA: Kluwer Academic Publishers, 1993.

[8] 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*, (Washington, D.C.), pp. 1–33, IEEE Computer Society Press, 1990.

[9] U. Bastola and G. Parisi, "The modular structure of Kauffman networks," *Phys. D*, vol. 115, p. 219, 1998.

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

[11] S. Bilke and F. Sjunnesson, "Stability of the Kauffman model," *Physical Review E*, vol. 65, p. 016129, 2001.

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

[13] J. Shelley, "Here we go again," 29 December 2004. http://www.gdnctr.com/dec_29_00.htm.

[14] E. Dubrova, M. Teslenko, and H. Tenhunen, "Computing attractors in dynamic networks," in *Proceedings of International Symposium on Applied Computing (IADIS'2005)*, (Algarve, Portugal), pp. 535–543, February 2005.

[15] E. Dubrova, "Modeling of gene regulatory systems by Random Boolean Networks," in *Microtechnologies for the New Millennium*, (Sevilla, Spain), May 2005.

[16] E. Dubrova and M. Teslenko, "Compositional properties of Random Boolean Networks," *Physical Review E*, 2005. to appear.

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

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

[19] V. G. Redko, "Kauffman's nk boolean networks," 1998. http://pespmc1.vub.ac.be/BOOLNETW.html.

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

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

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

[23] U. Bastola and G. Parisi, "The critical line of Kauffman networks," *J. Theor. Biol.*, vol. 187, p. 117, 1997.

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

[25] 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*, vol. 261, pp. 91–103, 2000.

[26] 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*, vol. 141, pp. 211–245, 1989.

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

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

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

[30] R. Dawkins, *The Selfish Gene*. Oxford: Oxford University Press, 1989.

[31] J. Suurkula, "Over 95 percent of DNA has largely unknown function," 2004. http://www.psrast.org/junkdna.htm.

[32] R. E. Tarjan, "Depth-first search and linear graph algorithms," *SIAM Journal on Computing*, vol. 1, no. 2, pp. 146–160, 1972.

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

[34] 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*, vol. 13, pp. 401–442, April 1994.

[35] M. Molloy and B. Reed, "The size of the giant component of a random graph with a given degree sequence," *Combin. Probab. Comput.*, vol. 7, pp. 295–305, 1998.

[36] A. Biere, A. Cimatti, E. M. Clarke, and Y. Zhu, "Symbolic model checking without BDDs," in *5th International Conference on Tools and Algorithms for Construction and Analysis of Systems*, (Amsterdam, The Netherlands), pp. 193–207, March 1999.

[37] P. Abdula, P. Bjesse, and N. Een, "Symbolic reachability analysis based on SAT-solvers," in *Proceedings of the 6th International Conference on Tools and Algorithms for the Analysis and Construction of Systems (TACAS'2000)*, vol. 1785 of LNCS, Springer-Verlag, 2000.

[38] A. Kuehlmann, M. K. Ganai, and V. Paruthi, "Circuit-based Boolean reasoning," in *Proceedings of the 38th ACM/IEEE Design Automation Conference*, (Las Vegas, Nevada), pp. 232–237, June 2001.

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

[40] 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*, (San Francisco), pp. 414–419, June 1995.

[41] 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)*, (Chicago, IL), pp. 125–138, Springer-Verlag, July 2000.

[42] 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*, (Tahoe City, CA), pp. 3–35 – 3–44, May 1995.

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

[44] 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.