GMR Genetics and Molecular Research
Online Journal
ISSN:1676-5680

An efficient algorithm for finding attractors in synchronous Boolean networks with biochemical applications

D. Zheng[1,2], G. Yang[1], X. Li[1,2], Z. Wang[1] and W.N.N. Hung[3]

[1]School of Computer Science and Engineering,
University of Electronic Science and Technology of China, Chengdu, China
[2]Department of Electronic Engineering, University of California, Los Angeles, CA, USA
[3]Synopsys Inc., Mountain View, CA, USA

Corresponding authors: D. Zheng / G. Yang
E-mail: desheng619@gmail.com / guowu@uestc.edu.cn

Genet. Mol. Res. 12 (4): 4656-4666 (2013)
Received November 27, 2012
Accepted August 8, 2013
Published October 18, 2013
DOI http://dx.doi.org/10.4238/2013.October.18.4

ABSTRACT. Self-organized systems, genetic regulatory systems and other living systems can be modeled as synchronous Boolean networks with stable states, which are also called state-cycle attractors (SCAs). This paper summarizes three classes of SCAs and presents a new efficient binary decision diagram based algorithm to find all SCAs of synchronous Boolean networks. After comparison with the tool BooleNet, empirical experiments with biochemical systems demonstrated the feasibility and efficiency of our approach.

Key words: Genetic regulatory system; Synchronous Boolean network; State-cycle attractors; Binary decision diagram (BDD); Biochemical system

INTRODUCTION

Recently, Boolean networks (Glass, 1985) have been widely used in biochemical systems. A book by Kauffman (Kauffman, 1995) gives a detailed description of this phenomenon. Each cell in our body coordinates the activities of about 100,000 genes along with the enzymes and proteins they produce. We can model a genetic regulatory system as a Boolean network. The genes follow Boolean rules to activate (switch ON) and inhibit (switch OFF) the next genes according to the activities of their molecular inputs, which lead to a network that follows a trajectory in its state space. Ultimately, the trajectory converges onto a state-cycle attractor (SCA) around which the system will cycle persistently. Therefore, we can find all of the SCAs to understand a biochemical system deeply. Before calculating all of the SCAs, we need to model a genetic regulatory system as a Boolean network. The Boolean network consists of a set of Boolean nodes or elements whose values are determined by other nodes or elements in the network. An element or node of a Boolean network has the value 1 (ON) or 0 (OFF) at any given time. These values are equivalent to active and inhibited elements of a genetic regulatory system.

There are many ways to model genetic regulatory systems, such as synchronous (Faure et al., 2006; Remy et al., 2006), asynchronous (Faure et al., 2006; Garg et al., 2008) and semi-asynchronous (Faure et al., 2006) models. Recently, Heidel (Heidel et al., 2003) and Farrow (Farrow et al., 2004) found the SCAs in synchronous Boolean networks. Zhao (Zhao, 2005) also proved the computing SCAs' methods in synchronous Boolean networks to be a nondeterministic polynomial time (NP) complete problem. Garg et al. (2008) proposed a resolution to calculate SCAs for synchronous and one class of asynchronous Boolean networks. Based on Garg's contribution, Ay et al. (2009) developed a faster method to search for the SCAs of self-loops and simple-loops. Dubrova (Dubrova et al., 2005) presented the basic principles of the tool BooleNet, which focuse on synchronous Boolean networks. Most of them used a binary decision diagram (BDD) (Lee, 1959) as the basic package to calculate the SCAs.

This paper summarizes three classes of SCAs and presents a new efficient BDD-based algorithm to find all SCAs of synchronous Boolean networks (FSSBN). The algorithm can calculate the number of SCAs and enumerate all of the states of the SCAs. When compared with BooleNet, experimental results show that our FSSBN algorithm is more efficient and feasible for large synchronous Boolean networks than pervious algorithms, and it can be used for real large biochemical system models.

The structure of this paper is organized as follows: in the next section 2, after a brief review of synchronous Boolean networks (SBNs), three classes of SCAs in SBNs are presented and defined. Then, we take Boolean models of two real biochemical systems as examples. In the following section, we present two properties and a theorem of SBNs. Then, we devise an algorithm, FSSBN, wherein we can find all of the SCAs and enumerate all of the states of the SCAs. In the results section, we use two experimental benchmarks to demonstrate that our approach is highly efficient and feasible. All of these benchmarks are from realistic biochemical systems. The final section concludes the paper and touches on future work.

MATERIAL AND METHODS

Three classes of SCAs for SBNs

In this section, we give a brief review of the SBNs and define three classes of SCAs

in SBNs. Then, we take Boolean models of two real biochemical systems as examples. *Synchronous Boolean networks*.

An SBN is a set of *n* nodes ($node_1$, $node_2$, $node_3$,..., $node_n$) which interact with each other in a synchronous manner (Farrow et al., 2004). At each given time $t \in N$ each node has only one of two different values: 1 (ON) or 0 (OFF). Thus, the synchronous Boolean network can be described as a set of equations.

$$node_{1,t+1} = f_1(node_{1,t}, node_{2,t}, node_{3,t}, \ldots, node_{n,t});$$
$$node_{2,t+1} = f_2(node_{1,t}, node_{2,t}, node_{3,t}, \ldots, node_{n,t});$$
$$node_{3,t+1} = f_3(node_{1,t}, node_{2,t}, node_{3,t}, \ldots, node_{n,t});$$
$$\ldots$$
$$node_{n,t+1} = f_n(node_{1,t}, node_{2,t}, node_{3,t}, \ldots, node_{n,t});$$

It can also be described in a simplified form.

$$NODE_{n,t+1} = f(NODE_t);$$

where nodes ($node_{1,t}$, $node_{2,t}$, $node_{3,t}$,..., $node_{n,t}$) are Boolean values at time *t*, $NODE_t$ = ($node_{1,t}$, $node_{2,t}$, $node_{3,t}$,…, $node_{n,t}$), $f = (f_1, f_2, f_3, \ldots, f_n)$ is the Boolean function from $\{0,1\}^n$ to $\{0,1\}^n$, which will transition the Boolean value of n nodes at time *t* to their Boolean values at time t + 1.

Basic definition

Given an SBN with *n* nodes and its Boolean function f, the Boolean value of $NODE_t$ at time *t* is called a state. One computation by *f* is called a step. *S* is the set with 2^n different states.

Definition 1. *Predecessor*: Given an SBN with *n* nodes and a Boolean function *f*, state s_j is the predecessor of state s_i, if $s_i = f(s_j)$, where $s_i, s_j \in S$, $1 \leq i, j \leq 2^n$. We use $((s_i^{pre})s_j, s_i)$ to represent that s_i^{pre} is the predecessor state of s_i.

Definition 2. *Successor*: Given an SBN with *n* nodes and a Boolean function *f*, state s_i is the successor of state s_j, if $s_i = f(s_j)$, where $s_i, s_j \in S$, $1 \leq i, j \leq 2^n$. We use $(s_i, s_j(s_i^{succ}))$ to represent that s_i^{succ} is the successor state of s_i.

Remark 1. Given an SBN with *n* nodes, for all the states, if we choose an input state, there will be only one determined output state by the Boolean function $f(input\_state) = output\_state$, where $input\_state$ and $output\_state$ represent the state at time *t* and *t* + 1, respectively. Just like Figure 1, there is only one $output\_state$ after each $input\_state$. However, if given an $output\_state$, there is at least one $input\_state$ that will reach it, shown in Figure 2.

Figure 1. One input state determined one output state in synchromous Boolean network.

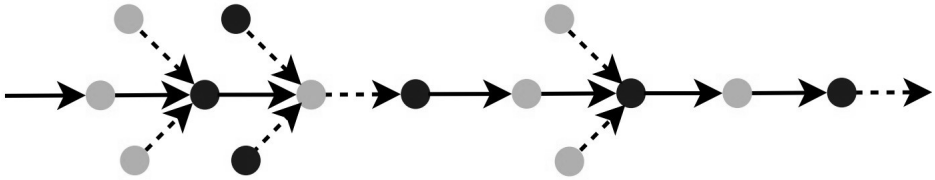

Figure 2. One output state exist at least one input state in synchromous Boolean network.

Definition 3. *Path*: Given an SBN with *n* nodes, we call $(s_1, s_2(s_1^{succ}), s_3(s_2^{succ}), \ldots, s_i(s_{i-1}^{succ}))$ a path, where $s_j = s_{j-1}^{succ}, 1 < j \leq i, s_i \in S, 1 \leq i \leq 2^n$.

Definition 4. *SCA*: Given an SBN with *n* nodes, the path $(s_i, s_{i+1}, s_{i+2}, s_{i+3}, \ldots, s_{i+n})$ is called an SCA, where all states in the path are different except state $s_{i+n+1}(s_{i+n}^{succ})$ and s_i, $i, n \in N$. We use SCA to represent an SCA in an SBN. The state number of an SCA is Length (SCA).

Remark 2. Given an SBN with *n* nodes, all SCAs are paths, but a path is not necessarily an SCA. Therefore, $SCAs \subseteq Paths$.

Three classes of SCAs

There are many kinds of SCAs in an SBN. Based on the nature of the SCA, SCAs can be categorized into one of three classes as follows:

Definition 5. *SCA without a branch*: Given an SCA of an SBN, if every state in the SCA has only one predecessor, it is an SCA without a branch, as shown in Figure 3A.

Definition 6. *SCA with some branches*: Given an SCA of an SBN, if some (but not all) states in the SCA have at least two predecessors, where $\forall s \in S', S' \subset SCA$, it is an SCA with some branches, as shown in Figure 3B.

Definition 7. *SCA with all branches*: Given an SCA of an SBN, if every state *s* in the SCA has at least two predecessors, it is an SCA with all branches, as show in Figure 3C.

We use two realistic biochemical system examples from previous reports (Robert, 1986; Heidel et al., 2003) to analyze the conditions of the SCAs. To begin, we use a simple affine system (line terms and constant terms) (Milligan and Wilson, 1993; Heidel et al., 2003) as an example. After reduction, the Boolean logic functions for this system are as follows. In this simple affine system, there are three Boolean nodes: A, B and C. Thus, the system has 23 different states. The visualized expressions of this example are shown in Figure 4A. The network contains two independent SCAs, and each state has only one predecessor state. Obviously, this SCA satisfies Definition 5 and is an SCA without a branch. Meanwhile, it is hard to resolve the large Boolean networks in this case. We will prove it in Section 3.

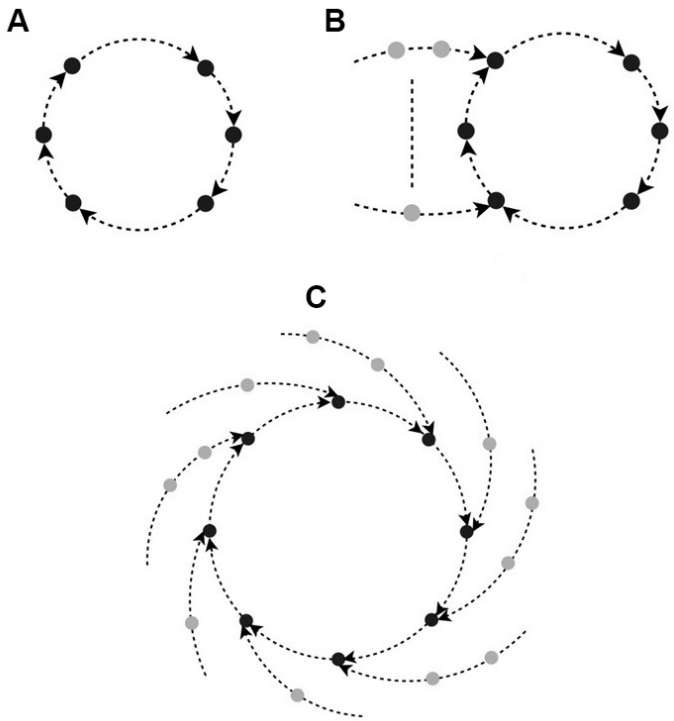

Figure 3. Model of each kind of state-cycle attractor (SCA). **A.** Model of SCA without branch; **B.** model of SCA with some branches; **C.** model of SCA with all branches.

$$A_{t+1} = 1 + C_t;$$
$$B_{t+1} = A_t;$$
$$C_{t+1} = B_t;$$

The second example is found in sigmoidal kinetics systems (Glass, 1973; Heidel et al., 2003). We chose one condition from the complicated environment with the following differential equations:

$$\frac{dA}{dt} = \frac{B^2}{1+B^2} \cdot \frac{D^2}{1+D^2} - A;$$
$$\frac{dB}{dt} = \frac{A^2}{1+A^2} - B;$$
$$\frac{dC}{dt} = \frac{1}{1+B^2} - C;$$

After simplification of the above differential equations, we can get the Boolean logic functions (Heidel et al., 2003). This is a nonlinear logic function with negative feedback.

$$A_{t+1} = B_t C_t;$$
$$B_{t+1} = A_t;$$
$$C_{t+1} = 1 + B_t;$$

This example shows that there are also three Boolean nodes: A, B, and C. According to the Boolean functions, we can get two independent SCAs (0,1,1)→(1,0,0)→(0,1,1) and (0,0,1)→(0,0,1), as shown in Figure 4B. The right SCA is a special case of Definition 7. The left SCA is suitable for Definition 6.

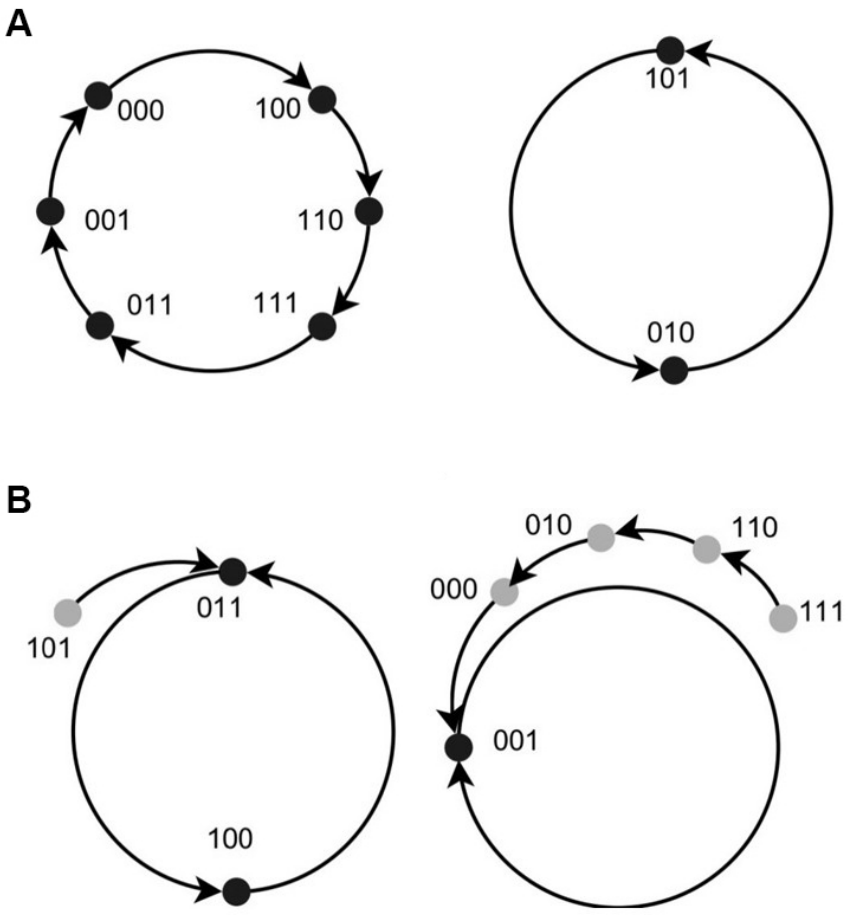

Figure 4. Example of each kind of state-cycle attractor (SCA). **A.** Example of SCA without branch; **B.** example of SCA with branches.

The two above examples show that an SBN generally contains two or three kinds of SCAs mixed together. Suppose there are three different SBNs with the same SCA length. The SCAs of each SBN fall into one class of SCAs. We know that when the SCAs fall under the model of Figure 3C, it is the easiest to resolve, whereas falling under Figure 3B is the second most difficult to resolve, and falling under the model of Figure 3A is the most difficult to resolve. We will prove this in the next section.

Finding SCAs

In this section, we first prove the main properties of an SBN. Then, we present our algorithm: finding SCAs in synchronous Boolean networks (FSSBN). Lastly, we use a theorem to demonstrate which kind of SCA is easy to resolve.

Theoretical result

Property 1. Given an SBN with n nodes, $s_i \in S, 1 \leq i \leq 2^n, n \in N$. If we start from s_i, we will get to an SCA before 2^n steps by the Boolean function $f(input\_state) = output\_state$.

Proof: Given an input state and applying it to the Boolean function $f(input\_state) = output\_state$, there will be 2^{n+1} states after 2^n steps. However, there are 2^n different states. Applying the pigeonhole principle, an SCA will be reached.

Property 2. Given an SBN with n nodes where $n \in N$, for all of the states of the Boolean network, the network contains at least one SCA.

Proof: According to Property 1, if we choose one state $s_i \in S, 1 \leq i \leq 2^n$, it will reach an SCA. Thus, there exists at least one SCA.

FSSBN algorithm

We used BDD to find SCAs in SBNs. First, we imported the Boolean logic function model or formula, which corresponds to the structure of the SBN. Using on the model, we can set up the BDD functions for the transition relations. We then initialized all variables, as shown in Step 1 of Algorithm 1. The integer variable *SCA_num* saves the number of SCAs. The BDD variable *current_set* stores all of the states that are not past. The BDD variable *current_path* records the current passed state. The BDD variable *tmp_set* covers all of the states that can reach the *current_path*. The BDD variables *s_input* and *s_output* are the input state and output state, respectively. The BDD variable *travel_state* covers all of the passed states. The function *Choose_A_State()* will pick a state from *current_set*. We can enumerate the states of SCAs using the *Print()* function. According to Property 2, given a Boolean function of an SBN, there is at least one SCA. Hence, if we start from any random state, we will eventually reach an SCA. Combining the above properties with BDD operations, we can compute the number of SCAs and enumerate them. The pseudo code is shown in Algorithm 1.

Theorem 1. Given three different synchronous Boolean networks with n nodes and their Boolean functions f_1, f_2 and f_3, their SCAs have the same length l_{sca}. The SCAs of f_1 fall under Definition 5 and are represented by $SCAs_{f_1}$. The SCAs of f_2 fall under Definition 6 and are represented by $SCAs_{f_2}$, in which the states of branches have only one predecessor and all of

the branches have a length l_{bra}. The SCAs of f_3 fall under Definition 7 and are represented by $SCAs_{f_3}$, in which the states of branches have only one predecessor and all of the branches have a length l_{bra}. The computing time will be $T(f_1(SCAs)) > T(f_2(SCAs)) > T(f_3(SCAs)$.

Proof: Suppose the function time of $f_{1,2,3}(input\_state, output\_state) = 1$ is T_f. There are 2^n different states. We analyzed the time of the above SBNs as a list according to Algorithm 1.

1. The number of $SCAs_{f_1}$ is $\left\lceil \frac{2^n}{l_{sca}} \right\rceil$. The computing time of one SCA is $l_{sca} \times T_f$. The total computing time is $\left\lceil \frac{2^n}{l_{sca}} \right\rceil \times l_{sca} \times T_f$.
2. Suppose that the $SCAs_{f_2}$ have *m* branches, where $0 < m < l_{sca}$. The worst computing time of the SCAs is $\left\lceil \frac{2^n}{l_{sca}+m\times l_{bra}} \right\rceil \times (l_{sca} + l_{bra} + 1) \times T_f$. The best computing time of the SCAs is $\left\lceil \frac{2^n}{l_{sca}+m\times l_{bra}} \right\rceil \times (l_{sca} + l_{bra}) \times T_f$. The average computing time of the SCAs is $\left\lceil \frac{2^n}{l_{sca}+m\times l_{bra}} \right\rceil \times (l_{sca} + l_{bra} + 0.5) \times T_f$.
3. The number of $SCAs_{f_3}$ is $\left\lceil \frac{2^n}{l_{sca}+l_{sca}\times l_{bra}} \right\rceil$. The worst computing time of the SCAs is $\left\lceil \frac{2^n}{l_{sca}+l_{sca}\times l_{bra}} \right\rceil \times (l_{sca} + l_{bra} + 1) \times T_f$. The best computing time of the SCAs is $\left\lceil \frac{2^n}{l_{sca}+l_{sca}\times l_{bra}} \right\rceil \times (l_{sca} + l_{bra}) \times T_f$. The average computing time of the SCAs is $\left\lceil \frac{2^n}{l_{sca}+l_{sca}\times l_{bra}} \right\rceil \times (l_{sca} + l_{bra} + 0.5) \times T_f$.

Then, we see that

$\left\lceil \frac{2^n}{l_{sca}} \right\rceil \times l_{sca} \times T_f > \left\lceil \frac{2^n}{l_{sca}+m\times l_{bra}} \right\rceil \times (l_{sca} + l_{bra} + 0.5) \times T_f > \left\lceil \frac{2^n}{l_{sca}+l_{sca}\times l_{bra}} \right\rceil \times (l_{sca} + l_{bra} + 0.5) \times T_f \rightarrow T(f_1(SCAs)) > T(f_2(SCAs)) > T(f_3(SCAs)$.

Algorithm 1. Finding SCAs in SBNs using BDD (FSSBN)

Input: Import the formula of SBNs, including *n* Boolean nodes. According to the formula, setup the BDD function $f(input\_state, output\_state) = 1$ and its BDD set $S = \{s_1, s_2, s_3, \ldots, s_{2^n}, \}$.

1: ***Initial*:**
int SCAs_num = 0;
BDD current_set = S;
BDD current_path = $\emptyset$;
BDD tmp_set = $\emptyset$;
BDD s_input = $\emptyset$; *s_output* = $\emptyset$;
2: ***while***(*current_set*≠$\emptyset$)***do***
3: *s_input = Choose_A_State* (*current_set*);
4: *current_path = current_path + s_output*;

5: *f*(*s_input,s_output*) = *1*;
6: ***while***((*current_path*∩*s_output*)== ∅)***do***
7: *current_path* = *current_path* + *s_output*;
8: *s_input* = *s_output*;
9: *f*(*s_input*, *s_output*) = *1*;
10: ***end while***
11: ***Print*** (*s_output*, *f*);
12: //*Find all the reachable states to current_path set*
13: *f*(*tmp_set*, *current_path*) = *1*;
14: ***while***((*tmp_set*−*current_path*)≠∅)***do***
15: *current_path* = *current_path* + *tmp_set*;
16: *f*(*tmp_set,current_path)* = *1*;
17: ***end while***
18: *SCAs_num*+ +;
19: *current_set* = *currentset* - *currentpath*;
20: *current_path* = ∅;
21: *tmp_set* = ∅;
22: *s_input* = ∅;
23: *s_output* = ∅;
24: ***end while***
25: ***Output*: *return*** *SCAs_num*

RESULTS AND DISCUSSION

In this section, we performed some experiments and compared the results of our algorithm to the results of BooleNet (Dubrova et al., 2005). The results show that our algorithm is efficient and highly feasible. We divided our experiments into two parts. The first part contains seven Boolean models of realistic organisms. The second part is a simple example that was previously published (Heidel, et al., 2003; Farrow et al., 2004. We generated some random benchmarks according to its features. All experiments are performed on an Intel Core™ CPU 4300 1.80 GHz with 2 gigabytes memory on an Ubuntu 9.04 Linux server.

Seven classical models

In this subsection, we used seven classical SBNs models of real organisms: *Arabidopsis thaliana* (Chaos et al., 2006), budding yeast (Li et al., 2004), *Drosophila melanogaster* (Albert and Othmer, 2003), fission yeast (Davidich and Bornholdt, 2008), mammalian cell (Faure et al., 2006), T-cell receptor (Klamt et al., 2006) and T-helper cell (Luis and Ioannis, 2006). The experimental results are shown in Table 1.

Table 1 shows the runtime of Algorithm 1 and BooleNet (Dubrova, et al., 2005) using the same conditions. “Timeout” represents the running time over 12x3600 = 43,200 s. “E” indicates that no result was returned. According to the results, we know that the number of SCAs is low. Almost all of the SCAs have some branches; therefore, it is easy to calculate. For *D. melanogaster* (Alber and Othmer, 2003), our routine wastes too much time to set up its Boolean functions. As the experimental results show, Algorithm 1 computes faster and resolves larger SBNs than BooleNet.

Table 1. Experiment results of seven classical models.

Benchmark	Nodes	Time (s)		SCA No.	
		BooleNet (Dubrova et al., 2005)	FSSBN	BooleNet (Dubrova et al., 2005)	FSSBN
Arabidopsis thaliana (Chaos et al., 2006)	15	0.024	0.023	10	10
Budding yeast (Li et al., 2004)	12	0.042	0.009	7	7
Drosophila melanogaster (Albert and Othmer, 2003)	52	Timeout	1576	E	7
Fission yeast (Davidich and Bornholdt, 2008)	10	0.044	0.009	13	13
Mammalian cell (Faure et al., 2006)	10	0.019	0.010	2	2
T-cell receptor (Klamt et al., 2006)	40	0.036	0.018	9	9
T-helper cell (Luis and Ioannis, 2006)	23	0.045	0.009	3	3

Random benchmarks

The experimental results of the previous section show that the number of SCAs is too small. Therefore, in this subsection, we will use a real organism model of protein-protein interactions (Gonze and Goldbeter, 2001; Heidel, et al., 2003) to extend our benchmarks. This is a model of phosphorylation/DE phosphorylation cycles. The Boolean logic function equations are the following. There are A, B, C, and D Boolean nodes. Each node can only be determined by the next one's current value. This is a feature that is very easy to obtain. It can be extended to any number of nodes.

$$A_{t+1} = B_t;$$
$$B_{t+1} = C_t;$$
$$C_{t+1} = D_t;$$
$$D_{t+1} = A_t;$$

Table 2 shows the protein experimental results. The SCAs of the protein network fall under Definition 5. Therefore, every state in the protein network Boolean model is in an SCA. The number of SCAs will increase with more Boolean nodes. The experimental results show that our algorithm is more efficient than BooleNet.

Table 2. Experiment results of protein.

Benchmark	Nodes	Time (s)		SCA No.	
		BooleNet (Dubrova et al., 2005)	FSSBN	BooleNet (Dubrova et al., 2005)	FSSBN
Protein					
	4	0.005	0.002	6	6
	9	0.040	0.013	60	60
	18	14.231	4.714	14602	14602
	27	Timeout	4453.169	E	4971068
	30	Timeout	39625.512	E	35792568
	33	Timeout	Timeout	E	E

In this paper, we applied BDD to SBNs of biochemical systems. We summarized and defined three classes of SCAs in SBNs. We listed two properties and one theorem of SBNs and

we devised an automatic algorithm (FSSBN) to find SCAs in SBNs. The experimental results demonstrate that the algorithm (FSSBN) can resolve large SBNs and calculate the number of SCAs in SBNs effectively.

ACKNOWLEDGMENTS

Research supported by the National Natural Science Foundation of China (Grant #60973016) and "973" Foundation (#2010CB328004).

REFERENCES

Albert R and Othmer HG (2003). The topology of the regulatory interactions predicts the expression pattern of the segment polarity genes in *Drosophila melanogaster*. *J. Theor. Biol.* 223: 1-18.

Ay F, Xu F and Kahveci T (2009). Scalable steady state analysis of Boolean biological regulatory networks. *PLoS One* 4: e7992.

Chaos A, Aldana M, Espinosa-Soto C, Leon B, et al. (2006). From genes to flower patterns and evolution: dynamic models of gene regulatory networks. *J. Plant Growth Regul.* 25: 278-289.

Davidich MI and Bornholdt S (2008). Boolean network model predicts cell cycle sequence of fission yeast. *PLoS One* 3: e1672.

Dubrova E, Teslenko M and Martinelli A (2005). Kauffman Networks: Analysis and Applications. In: Proceedings of the 2005 IEEE/ACM International Conference on Computer-Aided Design. *IEEE Comput. Soc.* 479-484.

Farrow C, Heidel J, Maloney J and Rogers J (2004). Scalar equations for synchronous Boolean networks with biological applications. *IEEE Trans. Neural Netw.* 15: 348-354.

Faure A, Naldi A, Chaouiya C and Thieffry D (2006). Dynamical analysis of a generic Boolean model for the control of the mammalian cell cycle. *Bioinformatics* 22: e124-e131.

Garg A, Di Cara A, Xenarios I, Mendoza L, et al. (2008). Synchronous versus asynchronous modeling of gene regulatory networks. *Bioinformatics* 24: 1917-1925.

Glass L (1985). Boolean and continuous models for the generation of biological rhythms. *Dyn. Syst. Cell. Automata* 197-206.

Glass L and Kauffman SA (1973). The logical analysis of continuous, non-linear biochemical control networks. *J. Theor. Biol.* 39: 103-129.

Gonze D and Goldbeter A (2001). A model for a network of phosphorylation-dephosphorylation cycles displaying the dynamics of dominoes and clocks. *J. Theor. Biol.* 210: 167-186.

Heidel J, Maloney J, Farrow C and Rogers J (2003). Finding cycles in synchronous Boolean networks with applications to biochemical systems. *Int. J. Bifurcat. Chaos* 13: 535-552.

Kauffman S (1995). At Home in the Universe. University Press New York, Oxford.

Klamt S, Saez-Rodriguez J, Lindquist JA, Simeoni L, et al. (2006). A methodology for the structural and functional analysis of signaling and regulatory networks. *BMC Bioinformatics* 7: 56.

Lee C (1959). Representation of switching circuits by binary-decision pro-grams. *Bell Syst. Tech. J.* 38: 985-999.

Li F, Long T, Lu Y, Ouyang Q, et al. (2004). The yeast cell-cycle network is robustly designed. *Proc. Natl. Acad. Sci. U. S. A.* 101: 4781-4786.

Luis M, and Ioannis X (2006). A method for the generation of standardized qualitative dynamical systems of regulatory networks. *Theor. Biol. Med. Model.* 3: 13.

Milligan D and Wilson M (1993). The behaviour of affine boolean sequential networks. *Connect. Sci.* 5: 153-167.

Remy E, Ruet P, Mendoza L, Thieffry D, et al. (2006). From logical regulatory graphs to standard petri nets: dynamical roles and func-tionality of feedback circuits. *Trans. Comput. Syst. Biol.* 7: 56-72.

Robert F (1986). Discrete Iterations: A Metric Study. Springer-Verlag, Berlin-Heidelberg-New York-Tokyo.

Zhao Q (2005). A remark on "scalar equations for synchronous Boolean networks with biological applications" by C. Farrow, J. Heidel, J. Maloney, and J. Rogers. *IEEE Trans. Neural Netw.* 16: 1715-1716.