An Algorithm to Find Cycles of Biochemical Systems

Desheng Zheng[1] Guowu Yang[1] Xiaoyu Li[1] Zhicai Wang[1]

[1]School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu 611731, China

Abstract

Genetic regulatory systems, self-organized systems and other living systems can be modeled as synchronous Boolean networks with stable states which are also called cycles. This paper devises two algorithms based on BDD to compute all the cycles in synchronous Boolean networks and enumerate all states in those cycles. Empirical experiments with biochemical systems demonstrate the feasibility and efficiency of our algorithms. It also shows that the two algorithms are conceptually so simple and efficient that they can be extensible to other realistic biochemical systems.

Keywords: genetic regulatory system, synchronous Boolean network, BDD biochemical system

1. Introduction

Recently, Boolean network [15] is widely used in biochemical systems. In [13], detailed description of this phenomenon is given. It introduces that each cell in our body coordinates the activities of about 100,000 genes and the enzymes and other proteins they produce. It can model a genetic regulatory system as a Boolean network. The genes follow Boolean rules and switch on and off according to the activities of their molecular inputs, which lead to a network following a trajectory in its state space. Ultimately, the trajectory converges onto a cycle which the system will cycle persistently thereafter. Therefore, cycles can help us to understand a biochemical system comprehensively. Synchronous Boolean network (SBN) is one kind of Boolean network, which all the nodes interact with each other in a synchronous manner. The problem of finding cycles in synchronous Boolean networks has been studied by [11][8][16], respectively.

A binary decision diagram (BDD) [14] is a data structure that is used to represent a Boolean function in circuits. BDDs are introduced by Lee [14], and further studied and made known by Akers [2], Boute [3] and Bryant [4]. Now there are many packages to support BDDs application, such as, ABCD [1], BuDDy [5], CMU BDD [6] and CUDD [7]. Based on our former research [12], we can apply it on biochemical systems flexible.

The structure of this paper is organized as follows. Section II gives some definitions and theorems. Section III presents two automatic algorithms: Finding Cycles in Synchronous Boolean Networks (FCSBN) and Optimized Finding Cycles in Synchronous Boolean Networks (OFCSBN), which can find and enumerate all cycles in a synchronous Boolean network. Section IV gives the application of three known examples of biochemical systems and experimental results. Section V concludes this paper and touch on future work.

2. Definitions and Theorems

Given a synchronous Boolean network with n nodes and its Boolean function f, the Boolean value at time t is called a state. The Boolean variable is also equivalent to the Boolean node. One computation by f is called a step. S is the set with 2^n different states.

Definition 1. Predecessor: Given a synchronous Boolean network 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 $s_i, s_j \in S, 1 \le i, j \le 2^n$.

Definition 2. Successor: Given a synchronous Boolean network with n nodes and a Boolean function f, state s_i is the predecessor of state s_j, if $s_i = f(s_j)$,where $s_i, s_j \in S, 1 \le i, j \le 2^n$. We use (s_i, s_i^S) to represent that s_i^S is one successor state of s_i.

Definition 3. Path: Given a synchronous Boolean network with n nodes, we call $(s_1, s_2(s_1^S), s_3(s_2^S), \ldots, s_i(s_{i-1}^S))$ as a path, where $s_j = s_{j-1}^S, 1 < j \le i, s_i \in S, 1 \le i \le 2^n$.

Definition 4. Cycle: Given a synchronous Boolean network with n nodes, the path $(s_i, s_{i+1}, s_{i+2}, s_{i+3}, \ldots, s_{i+n}, s_{i+n+1})$ is called a cycle, where all the states in the path are different except states s_{i+n+1} and s_i. We use C to represent a cycle in synchronous Boolean network. The state number of a cycle is $Num(C)$.

Theorem 1. Given a synchronous Boolean network with n nodes, $s_i \in S, 1 \le i \le 2^n, n \in N$. If we start from s_i, we will get to a cycle before 2^n steps by Boolean function *f(input_state) =output_state*.

Proof: Given an input state, applying to Boolean function *f(input_state) = output_state*, after 2^n steps, there will be 2^{n+1} states. However there are 2^n different states. Applying the Pigeonhole Principle, it will reach a cycle.

Theorem 2. Given a synchronous Boolean network with n nodes, $n \in N$, for all the states of Boolean network, the network contains at least one cycle.

Proof: According to Theorem 1, if we choose one state $s_i \in S, 1 \le i \le 2^n$, it will get to a cycle. Thus, it exist at least one cycle. □

3. Fcsbn And Ofcsbn

We use BDD to find cycles in synchronous Boolean networks. First, we need to import synchronous Boolean network logical functions and setup their BDD functions for the transition relations. We then initialize all variables, as shown in Step 1 of Algorithm 1. We use Boolean flag *repeat* to avoid repeated computing. The integer variable *cycleNum* saves the number of cycles. The BDD variable *currentSet* stores all the states are not passed. The BDD variable *currentPath* records the current passed state. The BDD variable *travelState* covers all the passed states. Function *Choose_A_State()* will randomly pick a state from *currentSet*. We can enumerate the cycle using the *Print()* function. According to Theorem 2, given a Boolean function of a synchronous Boolean network, there is at least one cycle. Hence, if we start from any random state, we will eventually reach a cycle. The pseudo-code is shown in Algorithm 1.

However, Algorithm 1 suffers from a big problem: repeated computing. We find a better idea to optimize Algorithm 1 to avoid repeated computing. The basic concept is given as Algorithm 2, which has a dramatically improved than Algorithm 1. We will analysis the result in Section IV, and show the improvement by comparing the experiment results.

Algorithm 1 Finding Cycles in Synchronous Boolean Networks (FCSBN)

Input: Import a synchronous Boolean network formula with *n* Boolean nodes and setup its BDD translation function *f(input_state)= output_state*.

```
1: Initial:
   bool repeat=FALSE;
   int cycleNum= 0;
   BDD currentSet=S;
   BDD currentPath= ø;
   BDD travelSet= ø;
   BDD s'= ø;
2: while(currentSet≠ø) do
3:   s'=Choose_A_State(currentSet);
4:   currentPath=currentPath+s';
5:   while((currentPath ∩f(s')) == ø) do
6:     if ((travelSet ∩f(s'))≠ø) then
7:       repeat=TRUE;
8:       break;
9:     end if
10:    s'=f(s');
11:    currentPath=currentPath+s';
12:  end while
13:  if (repeat≠TRUE) then
14:    cycleNum++;
15:  end if
16:  travelSet=travelSet+currentPath;
17:  currentSet=currentSet−travelSet;
18:  Print(currentPath);
19:  repeat=FALSE;
20:  currentPath= ø;
21:  s'= ø;
22: end while
Output:
23: return cycleNum;
```

In step 1 of Algorithm 2, there are some variable changed than Algorithm 1. We need to state them clearly. BDD variable *tmpSet* covers all the states can reach to the *currentPath*. BDD variable *s'* and *s''* are the input state and output state, respectively. The two functions *Choose_A_State()* and *Print()* are also same with the same function in the Algorithm 1.

Algorithm 2 Optimized Finding Cycles in Synchronous Boolean Networks (OFCSBN)

Input: Import a synchronous Boolean network formula with *n* Boolean nodes and setup its BDD translation function *f(input_state, output_state)=1*.

```
1: Initial:
   int cycleNum= 0;
   BDD currentSet=S;
   BDD currentPath= ø;
   BDD tmpSet= ø;
   BDD s'= ø; s''= ø;
2: while(currentSet≠ø) do
3:   s''=Choose_A_State(currentSet);
4:   currentPath=currentPath+s';
5:   f(s', s'') = 1;
6:   while((currentPath ∩s'') == ø) do
7:     currentPath=currentPath+s'';
8:     s'=s'';
9:     f(s', s'') = 1;
10:  end while
11:  Print(currentPath);
12:  f(tmpSet,currentPath) = 1;
13:  while((tmpSet−currentPath)≠ø) do
14:    currentPath=currentPath+tmpSet;
15:    f(tmpSet,currentPath) = 1;
16:  end while
17:  cycleNum++;
18:  currentSet=currentSet−currentPath;
19:  currentPath= ø;
20:  tmpSet= ø;
21:  s'= ø;
22:  s''= ø;
23: end while
Output:
24: return cycleNum;
```

4. Experimental Results

In this subsection, we conduct experiments based on Molecular [10], Protein [9] and Oscillation [11]. The benchmarks are constructed based on [11] [8], which generate different synchronous Boolean networks with scalar equations according to biochemical systems.

All experiments are performed on Intel® Core TM CPU 4300 1.80GHz with 2G memory on Ubuntu 9.04 Linux server. Table I shows the runtime of both algorithms under the same conditions and compares the number of cycles between algorithm FCSBN, OFCSBN and [11]. The Molecular and the Oscillation examples contain many cycles with branches. However, the Protein example contains cycles without branches. *Timeout* represents the running time over 10^5 seconds. *Unavailable* represents no corresponding cycle number in [11]. Experiments show that our algorithm is more feasible and efficiency.

Table I Experimental Results

Benchmark	Boolean Node Number	Time (Second)		Cycle Number	
		FCSBN	OFCSBN	OFCSBN	[16]
Molecular	6	0.008	0.004	2	2
	9	0.017	0.005	2	Unavailable
	18	14.104	0.005	2	Unavailable
	21	135.145	0.006	2	Unavailable
	24	1114.233	0.006	2	Unavailable
	27	7953.365	0.007	2	Unavailable
	30	75881.712	0.007	2	Unavailable
	402	Timeout	0.389	2	Unavailable
	1002	Timeout	3.091	2	Unavailable
Protein	4	0.006	0.005	6	6
	9	0.009	0.008	60	Unavailable
	18	4.714	3.695	14602	Unavailable
	21	46.562	35.839	99880	Unavailable
	24	445.269	337.414	699252	Unavailable
	27	4453.169	3321.923	4971068	Unavailable
	30	39625.512	30407.416	35792568	Unavailable
	33	Timeout	Timeout	Unavailable	Unavailable
Oscillation	6	0.009	0.004	3	3
	9	0.027	0.005	7	Unavailable
	18	14.596	0.011	22	Unavailable
	21	142.281	0.013	29	Unavailable
	24	1009.073	0.016	37	Unavailable
	27	8775.574	0.020	46	Unavailable
	30	81495.409	0.025	56	Unavailable
	402	Timeout	219.641	9046	Unavailable
	1002	Timeout	11259.153	55946	Unavailable

5. Conclusion

This paper presents two algorithms to find cycles of biochemical systems modeled as synchronous Boolean networks. The experimental results demonstrate that computing the number of all cycles and enumerating every state in synchronous Boolean networks can be resolved effectively. It also shows the two algorithms can be extended to other realistic biochemical systems.

6. Acknowledgment

This work was supported by the National Nature Science Foundation of China (No. 60973016) and 973 Foundation (No. 2010CB328004).

7. References

[1] http://fmv.jku.at/abcd/.

[2] S.B. Akers. Binary decision diagrams. *IEEE Trans. on Computers*, 100(6): 509–516, 1978.

[3] R.T. Boute. The binary decision machine as programmable controller. *Euromicro Newsletter*, 2(1):16–22, 1976.

[4] R.E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Trans.onComputers*,100(8):677–91,1986.

[5] http://www.itu.dk/research/buddy

[6] http://www.cs.cmu.edu/modelcheck/bdd

[7] http://vlsi.colorado.edu/ fabio/cudd

[8] C. Farrow, J. Heidel, J. Maloney, and et al., Scalar equations for synchronous boolean networks with biological applications. *IEEE Trans. on Neural Networks*, 15(2):348–354, 2004.

[9] D. Gonze and A. Goldbeter. A model for a network of phosphorylation-dephosphorylation cycles displaying the dynamics of dominoes and clocks. *Journal of Theoretical Biology*, 210(2):167–186, 2001.

[10] B.C. Goodwin. Temporal organization in cells. Academic Press, 1963.

[11] J. Heidel, J. Maloney et al. Finding cycles in synchronous boolean networks with applications to biochemical systems. *International Journal of Bifurcation and Chaos in Applied Sciences and Engineering*, 13(3):535–552, 2003.

[12] W. Hung, X. Song, E.M. Boulhamid, Bdd minimization by scatter search. *IEEE Trans. on Computer-Aided*

Design of Integrated Circuits and Systems, 21(8):974–979, 2002.

[13] S.A. Kauffman. At home in the universe. *Oxford University Press New York*, 1995.

[14] C.Y. Lee. Representation of switching circuits by binary decision programs. *Bell System Technical Journal*, 38(4):985–999, 1959.

[15] R.J. Tocci. Digital systems: principles and applications. Pearson Education India, 1988.

[16] Q. Zhao. A remark on "scalar equations for synchronous boolean networks with biological applications by C. Farrow, J. Heidel, J. Maloney, and J. Rogers". *IEEE Trans. on Neural Networks*, 16(6): 1715–1716, 2005.