An FVS-based Approach to Attractor Detection in Asynchronous Random Boolean Networks

Trinh Van Giang, Tatsuya Akutsu, *Senior Member, IEEE*, and Kunihiko Hiraishi, *Member, IEEE*

Abstract—Boolean networks (BNs) play a crucial role in modeling and analyzing biological systems. One of the central issues in the analysis of BNs is attractor detection, i.e., identification of all possible attractors. This problem becomes more challenging for large asynchronous random Boolean networks (ARBNs) because of the asynchronous and non-deterministic updating scheme. In this paper, we present and formally prove several relations between feedback vertex sets (FVSs) and dynamics of BNs. From these relations, we propose an FVS-based method for detecting attractors in ARBNs. Our approach relies on the principle of removing arcs in the state transition graph to get a candidate set and the reachability property to filter the candidate set. We formally prove the correctness of our method and show its efficiency by conducting experiments on real biological networks and randomly generated N-K networks. The obtained results are very promising since our method can handle large networks whose sizes are up to 101 without using any network reduction technique.

Keywords—feedback vertex sets, asynchronous random Boolean networks, attractors, reachability analysis, binary decision diagrams, Petri nets, unfoldings

1 Introduction

Boolean networks (BNs) play a crucial role in modeling and analysis of complex biological networks (e.g., gene regulatory networks [1]). They have also been applied to many areas beyond systems biology, such as, mathematics, neural networks, social modeling, and robotics (see [2]). Many different types of updating schemes of BNs (synchronous, asynchronous, deterministic, or non-deterministic), which regulate the way the nodes are updated through time evolution, have been proposed and widely studied. The results of [3], [2], [4] show that different types of updating schemes produce different behaviors of the same BN.

In the landscape of dynamics of a dynamical system, we can distinguish between the transient and long-run dynamics. In BNs or other qualitative models, the long-run dynamics are referred to as *attractors*. An attractor of a BN is a set of states such that the BN can not escape from this set once entered it. In the biological context, attractors of a BN are linked to phenotypes [5] or functional cellular states (e.g., proliferation, apoptosis, or differentiation) [6]. Thus, analysis of attractors could provide new insights into systems biology (e.g., the origins of cancer [7]). Moreover, attractors also play an important role in the development of new drugs [8]. Therefore, attractor detection is of great importance in analyzing biological systems modeled as BNs.

In practice, attractor detection of a BN can become very expensive as the size n of the BN grows since the number of possible states of a BN is exponential in the number of nodes. Many algorithms and tools have been developed in the efforts to efficiently solve this problem. For small networks (e.g., $n \leq 50$), attractors can be simply detected by various enumeration and simulation methods [9], [10], [11]. For larger networks, attractors can be efficiently detected with two techniques including binary decision diagram (BDD) and satisfiability (SAT) solvers. In BDD-based methods [12], [13], the transition relation of a BN is encoded with BDD and the calculation of attractors exploits advantages of the efficient BDD operations. However, these methods still rely on the exhaustive traversal of the whole state space, making their efficiency is strictly prevented when the BN becomes large, e.g., n is over 100. SAT-based methods [14] encode the attractor detection problem as a satisfiability problem, then exploit the efficient implementation of SAT solvers. They can handle larger networks within shorter time compared to BDD-based methods. There are also some methods [15], [16], [11] exploiting the relations between network structure and network dynamics.

The above-mentioned methods are mainly designed for synchronous BNs in which all the nodes will be updated simultaneously. In biology, the updating process of each gene may spend various time from fractions of a second to hours [4]. With the need to represent various time scales, asynchronous BNs are considered more realistic [17]. In this paper, we focus on asynchronous random Boolean networks (ARBNs) [18], [4] where at each time step a randomly selected node is updated and there is no restriction on Boolean functions.

Attractor detection in ARBNs becomes more challenging especially for large networks because of the asynchronous and non-deterministic updating scheme. There are two main types of attractors [13]: singleton attractors and cyclic attractors. A singleton attractor is formed by only one state. A cyclic attractor is formed by one or more overlapping cycles of states. In synchronous BNs, an attractor is either a singleton state or a cycle since each state has exactly one outgoing transition. Under the asynchronous and non-deterministic updating scheme, each state may have multiple outgoing transitions. Therefore, an attractor of an ARBN, in general, can be seen as a terminal strongly connected component in the state transition graph of this ARBN. This complex attractor structure makes SAT-based methods ineffective since the respective SAT formulas become

• *T. Van Giang is with the School of Information Science, Japan Advanced Institute of Science and Technology, Nomi 923-1292, Japan (e-mail: giang-trinh@jaist.ac.jp).*
T. Akutsu is with the Bioinformatics Center, Institute for Chemical Research, Kyoto University, Kyoto 611-0011, Japan (e-mail: takutsu@kuicr.kyoto-u.ac.jp).
K. Hiraishi is with the School of Information Science, Japan Advanced Institute of Science and Technology, Nomi 923-1292, Japan (e-mail: hira@jaist.ac.jp).

Manuscript received April 19, 2005; revised January 11, 2007.

prohibitively large.

A few methods have been proposed in the efforts to efficiently solve attractor detection in ARBNs. The first effort is the BDD-based method [12] as an extension of that for synchronous BNs. The authors then improved this method by exploiting the relations between attractors of an ARBN and attractors of its synchronous counterpart [13]. Their implemented tool (called genYsis) has also been used widely in the research communities. In 2011, Skodawessely proposed a method [19] to detect ARBN attractors based on feedback vertex sets and the principle of reducing dynamics. This method seems to be promising but it can only handle the networks whose sizes are up to 38. Recently, a decomposition method [20] was proposed to deal with large BNs. This method decomposes a large ARBN into smaller components (called *blocks*) based on the network structure, detects attractors in these blocks, and then recovers the attractors of the original ARBN. Clearly, the efficiency of this method largely depends on the sizes of the decomposed blocks.

Inspired by the principle of reducing dynamics by [19], we propose a new method to handle attractor detection in asynchronous Boolean networks, especially for large ones. The main idea of our method is similar to the idea of [19]. However, we here present several generalized and improved results in both theoretical and practical aspects. We first present and prove several relations between feedback vertex sets (FVSs) and dynamics of BNs. From these relations, we propose an FVS-based method for detecting all possible attractors of an ARBN. Our approach relies on an FVS of this ARBN to get a candidate set of states such that each attractor of the ARBN contains at least one state of this set. We then filter the candidate set by checking reachability property in the ARBN. The obtained set one-to-one corresponds to the set of all attractors of the ARBN and is sufficient since starting from a state s in an attractor, we can enumerate this attractor by listing all states reachable from s. Our method includes several constituent steps. For each step, we formalize the corresponding problems, analyze them, and propose efficient solutions for them. The correctness of our method is formally proved. We also propose a preprocessing to reduce the computational burden while still preserving the correctness. The experimental results confirm the usefulness of the preprocessing and are very promising since our method can handle large networks whose size are up to 101 without using any network reduction technique.

The rest of this paper is organized as follows: Section 2 gives preliminaries on Boolean networks, attractors, feedback vertex sets, and Petri net unfoldings. Section 3 presents the relations between FVSs and BNs. Section 4 presents the FVS-based method for attractor detection in ARBNs and its constituent steps. Results of the experiments are shown in Section 5. Section 6 concludes the paper and discusses open problems.

2 Preliminaries

2.1 Boolean networks and attractors

A Boolean network $\mathcal{N} = (V, F)$ consists of a set of n nodes $V = \{x_1, x_2, ..., x_n\}$ and a set of n Boolean functions $F = \{f_1, f_2, ..., f_n\}$. In the context of gene regulatory networks, V and F correspond to a set of genes and a set of gene regulatory rules, respectively. The ith node is associated to a Boolean variable $x_i \in \{0, 1\}$ and a Boolean function $f_i : \{0, 1\}^{|IN(f_i)|} \rightarrow \{0, 1\}$ where $IN(f_i)$ denotes the set of input nodes of Boolean function f_i. Note that we use x_i to refer to the ith node and its associated Boolean variable. The *interaction graph* of a BN $\mathcal{N}$, denoted by $IG(\mathcal{N})$, is the directed graph defined as follows: the set of nodes is V and, for all $x_i, x_j \in V$ (not necessarily distinct), there is an arc (x_j, x_i) if $x_j \in IN(f_i)$.

A state of the network is given by a vector $x = (x_1, ..., x_n) \in \{0, 1\}^n$. At each time step, node x_i can update its value by $x'_i = f_i(x)$ where x is the current state of the BN and x'_i is the next value of x_i. For simplicity, we use the notation $f_i(x)$ even if $IN(f_i) \subset V$. An updating scheme specifies the way the nodes will be updated. The BN can transit from a state to a state based on its updating scheme. This is a *state transition*. Dynamics of a BN are captured by a state transition graph (STG). A STG is a directed graph whose nodes and arcs denote states and state transitions, respectively. In this paper, we use $x \xrightarrow{\mathcal{N}} x'$ for the transition (x, x') in the STG of $\mathcal{N}$. We write $\xrightarrow{\mathcal{N}}^*$ for the reflexive transitive closure of $\xrightarrow{\mathcal{N}}$. $FR^{\mathcal{N}}(S)$ denotes the set of states reachable from the states of the set S in the STG of $\mathcal{N}$. Formally, $FR^{\mathcal{N}}(S) = \{s' | s \xrightarrow{\mathcal{N}}^* s', s \in S\}$. We can also use $x \rightarrow x'$, $x \rightarrow^* x'$, or $FR(S)$ if the BN is already specified.

In general, a Boolean function can be formed by any combinations of any logical operators (e.g., AND $\wedge$, OR $\vee$, and NEGATION $\neg$) on variables associated with its input nodes. In this paper, we focus on random Boolean networks (i.e., there is no restriction on Boolean functions). There are two major types of random BNs [3] including Classical Random Boolean Networks (CRBNs) and Asynchronous Random Boolean Networks (ARBNs). CRBNs were proposed by Stuart Kauffman [21] to model gene regulatory networks in cells. They have a synchronous updating, i.e., all nodes at time $t + 1$ take into account nodes at time t for their updating. Since CRBNs are deterministic, the STG of a CRBN has 2^n nodes and 2^n arcs. ARBNs were firstly studied by Harvey and Bossomaier [18]. Their updating is asynchronous and non-deterministic. At each time step, a single node is selected at random in order to be updated. The STG of an ARBN has exactly 2^n nodes and may have up to $n \times 2^n$ arcs.

Attractors are a key behavior of a BN. The formal definition of an attractor is given in Definition 1. In the STG of a BN, an attractor is equivalent to a terminal strongly connected component. Since the STG of a BN has 2^n nodes and at least 2^n arcs, naive approaches for finding attractors (e.g, explicitly building the STG and then applying graph algorithms) are intractable when n is large.

Definition 1. *An attractor of a BN is a set att of states satisfying any state in att can reach any other state in att and can not reach any other state that is not in att.*

We can classify two main types of attractors: singleton and cyclic attractors. A singleton attractor (or a fixed point) has only one state. A cyclic attractor has at least two states and is formed by overlapping one or more cycles of states. A CRBN and its ARBN counterpart share the same set of singleton attractors. Singleton attractors can easily be identified by calculating the fixed points (i.e., the states satisfying $x = f(x)$). A cyclic attractor of a CRBN only consists of one cycle while a cyclic attractor of an ARBN can consist of at least two overlapping cycles. This makes identification of ARBN attractors more difficult.

We use the BN shown in Example 1 as a straight illustrative example used in the following sections. Figure 1 shows its interaction graph. Figure 2 shows two STGs of the CRBN counterpart and the ARBN counterpart of this BN. Herein, the

CRBN has one fixed point ($\{000\}$) and two cyclic attractors ($\{100, 011\}$ and $\{101, 111\}$). The ARBN has one fixed point ($\{000\}$) and one cyclic attractor ($\{101, 111\}$).

Example 1. *Consider a BN of three nodes (x_1, x_2, and x_3). Its Boolean functions are given by:*

$$\begin{cases} f_1 = x_2 \vee x_3, \\ f_2 = x_1 \wedge \neg x_2, \\ f_3 = x_1. \end{cases}$$

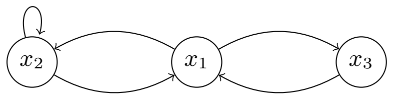


Fig. 1: Interaction graph of the BN in Example 1.

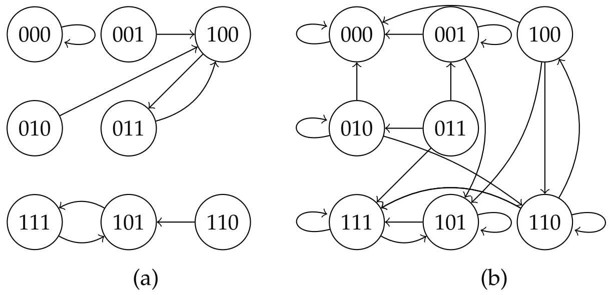


Fig. 2: STGs of (a) the CRBN and (b) the ARBN.

2.2 Feedback vertex sets

Feedback vertex set (FVS) is an important concept in graph theory. A feedback vertex set of a graph is a set of nodes such that removing them makes the graph acyclic. For example, the interaction graph of the BN of Example 1 has three possible FVSs including $\{x_1, x_2\}$, $\{x_2, x_3\}$, $\{x_1, x_2, x_3\}$. $\{x_1, x_2\}$ and $\{x_2, x_3\}$ are two minimum FVSs. The problem of finding a minimum feedback vertex set is known to be NP-hard [22]. Some approximation algorithms have been developed [23] to solve this problem. However, such algorithms are usually complicated. Thus, we here use a simple greedy algorithm for finding an (not necessarily minimum) FVS. This greedy algorithm relies on strongly connected components (SCCs).

We first recall some definitions on graph and an observation on FVSs. Let $IG(\mathcal{N}) = (V, E)$ be the interaction graph of the BN $\mathcal{N}$. $IG(\mathcal{N})$ is a directed graph. An SCC c is trivial if c is made of a single vertex v and $(v, v) \notin E$, and is non-trivial otherwise. A vertex v is a self vertex if $(v, v) \in E$. Note that if v is a self vertex, all FVSs of $IG(\mathcal{N})$ must contain v.

Our greedy algorithm is given in Algorithm 1. We here use Tarjan's algorithm [24] for finding the set of SCCs. Then, the set of non-trivial SCCs is easily obtained. $IG(\mathcal{N})[c]$ denotes the subgraph of $IG(\mathcal{N})$ induced by the set of vertices c. $IG(\mathcal{N}) - V_{self}$ is equivalent to $IG(\mathcal{N})[V \backslash V_{self}]$.

Algorithm 1 *Algorithm for finding a feedback vertex set*

```
Input: A directed graph $IG(\mathcal{N}) = (V, E)$
Output: A feedback vertex set $U$
 1: $U \leftarrow \emptyset$
 2: $V_{self} \leftarrow$ the set of self vertices of $IG(\mathcal{N})$
 3: $U \leftarrow U \cup V_{self}$
 4: $IG(\mathcal{N}) \leftarrow IG(\mathcal{N}) - V_{self}$
 5: $C \leftarrow$ the set of non-trivial SCCs of $IG(\mathcal{N})$
 6: while $C \neq \emptyset$ do
 7:     Randomly pick an SCC $c$ from $C$
 8:     Pick a vertex $v$ with the maximum outdegree from $c$
 9:     $U \leftarrow U \cup \{v\}$
10:     $C_c \leftarrow$ the set of non-trivial SCCs of $IG(\mathcal{N})[c \backslash \{v\}]$
11:     $C \leftarrow C \cup C_c$
12: end while
13: return $U$
```

2.3 Petri net unfoldings

Petri nets [25] are a basic model of parallel and distributed systems. A Petri net (PN) is a bipartite graph whose nodes are either places or transitions. In this paper, we only consider 1-safe Petri nets where the number of tokens of each place is either 0 (unmarked) or 1 (marked). The set of marked places forms a marking of the PN. If the PN has an arc (p, t), then p is called an input place of t. If the PN has an arc (t, p), then p is called an output place of t. A transition is enabled if all its input places are marked. When a transition is enabled, it can fire. The firing of this transition makes all its input places unmarked and then makes all its output places marked, modifying the current marking of the PN. Note that, when multiple transitions are enabled, only one transition can fire.

Formally, a 1-safe PN is a tuple $\mathcal{P} = \langle P, T, pre, post, M_0 \rangle$. P and T are sets of places and transitions, respectively. $pre \subseteq P \times T$ is the set of all arcs from places to transitions while $post \subseteq T \times P$ is the set of all arcs from transitions to places. For any place p, we say *pre-set* of p is the set $^\bullet p = \{t \in T | (t, p) \in post\}$ and *post-set* of p is the set $p^\bullet = \{t \in T | (p, t) \in pre\}$. For any transition t, we say *pre-set* of t is the set $^\bullet t = \{p \in P | (p, t) \in pre\}$ and *post-set* of t is the set $t^\bullet = \{p \in P | (t, p) \in post\}$. A subset $M \subseteq P$ of the places is called a marking. M_0 is the initial marking of the PN.

A transition t of a 1-safe PN is enabled at a marking M if and only if $^\bullet t \subseteq M$. The firing of t leads to a new marking M' specified by $M' = (M \backslash {}^\bullet t) \cup t^\bullet$. We denote this marking transition by $M \xrightarrow{t} M'$. A marking M' is called reachable from a marking M if there exists a firing sequence $w = t_1 t_2 ...$ over T such that $M \xrightarrow{t_1} M_1 \xrightarrow{t_2} M_2 ... \rightarrow M'$. A marking reachable from M_0 is called a *reachable marking* of the PN. All reachable markings of the PN and their marking transitions are represented by a directed graph called the *reachability graph*.

Figure 3a shows an example 1-safe PN. The places are represented by circles and the transitions are represented by squares. The arcs and the initial marking are represented by the arrows and dots in the marked places, respectively. Herein, $M_0 = \{p_1\}$. At this marking, t_1 and t_2 are enabled. Figure 3b shows the reachability graph of this PN.

Petri net unfoldings [26] aim at representing the reachability graph of a 1-safe Petri net by exploiting concurrency between transitions to prune redundant interleavings of these transitions. The unfolding of a 1-safe PN $\mathcal{P}$ can be seen as an acyclic Petri net $\mathcal{U}$ that has the same behaviors as $\mathcal{P}$. In general, $\mathcal{U}$

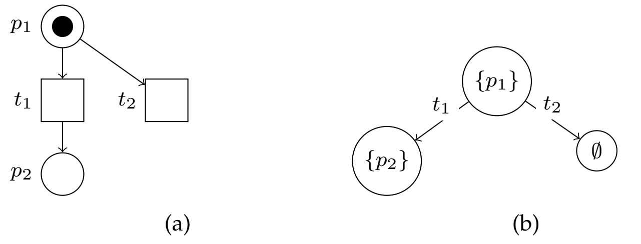


Fig. 3: (a) A 1-safe Petri net and (b) its reachability graph with the initial marking $\{p_1\}$. In (b), the text above each arrow denotes the fired transition.

is infinite. However, there exists a finite prefix $\mathcal{P}_\mathcal{U}$ of $\mathcal{U}$ since $\mathcal{P}$ is safe. Generally speaking, a finite prefix is an acyclic Petri net whose sets of places, transitions, and arcs are finite and are subsets of the sets of places, transitions, and arcs of $\mathcal{U}$, respectively. $\mathcal{P}_\mathcal{U}$ is complete since every reachable marking of $\mathcal{P}$ has a reachable counterpart in $\mathcal{P}_\mathcal{U}$. Thus, $\mathcal{P}_\mathcal{U}$ represents the set of reachable markings of $\mathcal{P}$. See [27] for more details on finite complex prefix. Regarding the reachability problem of 1-safe Petri nets, we can build $\mathcal{P}_\mathcal{U}$ and then easily check whether a marking M reaches a marking M'. This function is implemented in Mole [28] which is an efficient tool for checking reachability property of 1-safe Petri nets based on unfoldings. Mole also supports a function allowing that building an unfolding immediately stops when a specific transition t can be added to this unfolding.

3 Relations between FVSs and BNs

We formally formulate and prove the below lemmas and theorems on relations between dynamics of a BN and it feedback vertex sets.

Lemma 1. *Let $\mathcal{N}$ be a BN whose interaction graph is acyclic. Then the STG of $\mathcal{N}$ has no cycles.*

Proof: We prove this lemma by using induction on the size n of $\mathcal{N}$.

Let G be the STG of $\mathcal{N}$. The case $n = 1$ is trivial since G has clearly only fixed points. Assume that G has no cycles with $n = k$.

We consider the case $n = k + 1$. There exists a node x_i without incoming arcs since $IG(\mathcal{N})$ is acyclic. Then, x_i is fixed, i.e., $f_i = a_i, a_i \in \{0, 1\}$. In a cycle of G, the value of x_i must not be changed since x_i is always a_i once its value receives a_i. x_i can be either 0 or 1. Let $f_1^\mathcal{N}, ..., f_{k+1}^\mathcal{N}$ be Boolean functions of $\mathcal{N}$. Then, $\mathcal{N}_1 = (V^{\mathcal{N}_1}, F^{\mathcal{N}_1})$ and $\mathcal{N}_2 = (V^{\mathcal{N}_2}, F^{\mathcal{N}_2})$ be two BNs of k nodes where $V^{\mathcal{N}_1} = V^{\mathcal{N}_2} = V^\mathcal{N} \backslash \{x_i\}$, $f_j^{\mathcal{N}_1} = f_j^N(x_1, ..., x_i/a_i, x_{i+1}, ..., x_{k+1}), f_j^{\mathcal{N}_2} = f_j^N(x_1, ..., x_i/(1 - a_i), x_{i+1}, ..., x_{k+1})$ ($j \in \{1, ..., k + 1\}$, $j \neq i$). Let G_1 and G_2 be the STGs of $\mathcal{N}_1$ and $\mathcal{N}_2$, respectively. Obviously, a cycle of G corresponds to a cycle of either $\mathcal{N}_1$ or $\mathcal{N}_2$. Since the interaction graphs of $\mathcal{N}_1$ and $\mathcal{N}_2$ are acyclic and have k nodes, G_1 and G_2 have no cycles by the induction hypothesis. Therefore, G has no cycles.

□

Lemma 2. *Let $\mathcal{N}$ be a BN and its STG be G. Let U be an FVS of $\mathcal{N}$. Then G has no cycles such that the values of the nodes in U does not change through these cycles.*

Proof: We prove this lemma by contradiction and using Lemma 1.

Let n be the number of nodes of $\mathcal{N}$. Without loss of generality, we reorder the nodes of N such that $U = \{x_1, ..., x_k\}$ and $V^\mathcal{N} \backslash U = \{x_{k+1}, ..., x_n\}$.

Assume that G has a cycle such that the values of the nodes in U do not change through this cycle (1). That is the values of x_i ($i \in \{1, ..., k\}$) are fixed. Let $x_i = a_i, a_i \in \{0, 1\}$ ($i \in \{1, ..., k\}$). Then, $\mathcal{N}' = (V^{\mathcal{N}'}, F^{\mathcal{N}'})$ be a BN of $n - k$ nodes where $V^{\mathcal{N}'} = V^\mathcal{N} \backslash U$, $f_j^{\mathcal{N}'} = f_j^\mathcal{N}(x_1/a_1, ..., x_k/a_k, x_{k+1}, ..., x_n)$ ($j \in \{k + 1, ..., n\}$). Let G' be the STG of $\mathcal{N}'$. Obviously, G' has a cycle by (1). $IG(\mathcal{N}')$ is acyclic since U is an FVS. So, G' has no cycles by Lemma 1. This is a contradiction. Hence, G has no cycles such that the values of the nodes in U do not change through these cycles. □

Theorem 1. *Let $\mathcal{N}$ be a BN and its STG be G. Let $U = \{x_{i_1}, ..., x_{i_k}\}$ be an FVS of $\mathcal{N}$. Let $B = \{b_{i_1}, ..., b_{i_k}\}$ be the set of retained values corresponding to the nodes of U. G' is the graph obtained by removing all arcs (x, x') from G where $\bigvee_{j=1}^k (x_{i_j} \leftrightarrow b_{i_j} \wedge x'_{i_j} \leftrightarrow 1 - b_{i_j})$ (1) holds. That means an arc (x, x') will be removed if it changes at least one node $x_{i_j} \in U$ from b_{i_j} to $1 - b_{i_j}$. Then G' has no cycles.*

Proof: We prove this theorem by using Lemma 2 to show that all cycles of G disappear in G'.

Let c be an arbitrary cycle of G. By Lemma 2, there is a node $x_i \in \{x_{i_1}, ..., x_{i_k}\}$ such that x_i changes its value through this cycle. Since c is a cycle of states, it must contain an arc (x, x') such that $x_i = b_i$ and $x'_i = 1 - b_i$. (x, x') satisfies (1) and will be removed. Then, c will disappear in G'.

Since c is arbitrary, all cycles of G will disappear in G'. Hence, G' has no cycles. In other words, G' has only fixed points. □

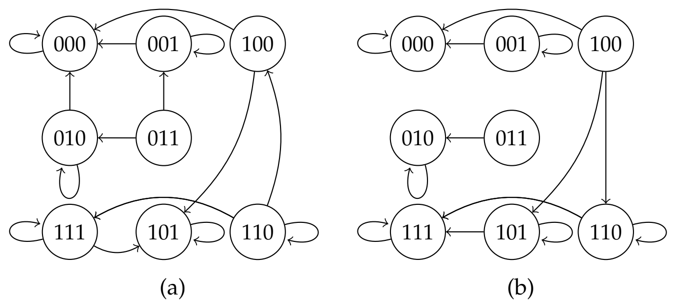


Fig. 4: The reduced STGs of the ARBN of the BN in Example 1 corresponding to (a) $U = \{x_1, x_2\}, b_1 = 0, b_2 = 0$ and (b) $U = \{x_1, x_2\}, b_1 = 0, b_2 = 1$.

Theorem 1 shows that the reduced STG of a BN has only fixed points. For example, let $\mathcal{N}$ be the ARBN of the BN in Example 1, G be as in Figure 2b, U be $\{x_1, x_2\}$. If $B = \{b_1, b_2\} = \{0, 0\}$, then G' is as in Figure 4a. The removed arcs are $(001, 101)$, $(010, 110)$, $(011, 111)$, $(100, 110)$, and $(101, 111)$. Obviously, G' has only fixed points.

Intuitively, the removal of arcs of the STG of the BN may only increase the number of attractors; each attractor of the original STG contains at least one attractor of the reduced STG. The relations between the original STG and the reduced STG are given in Lemma 3 and Theorem 2.

Lemma 3. *Let $\mathcal{N}$ be a BN and its STG be G. G' is the graph obtained by removing an arc (x, y) from G. Let A and A' be the sets* *of attractors of G and G', respectively. Then, there exists a mapping $m : A \to A'$ with $m(att) \subseteq att$ for all $att \in A$ and $m(att_1) \neq m(att_2)$ for all $att_1, att_2 \in A, att_1 \neq att_2$ (1).*

Proof: We consider all two cases as follows.

Case 1: (x, y) is not in any attractor of G. Obviously, all attractors of G are still in G'. Choose the mapping m such that $m(att) = att$ for all $att \in A$.

Case 2: (x, y) is in an attractor of G. Since attractors are pairwise disjoint, (x, y) is only in one attractor of G say att. Obviously, $x \in att$ and $y \in att$. Let F be the forward reachable set of x in G' (i.e., the set of all states reachable from x). Then $F \subseteq att$. There is an attractor att' of G' such that $att' \subseteq F$. So, $att' \subseteq att$. All attractors of $A \backslash \{att\}$ of G are still in G'. Choose the mapping m such that $m(s) = s$ for $s \in A \backslash \{att\}$ and $m(att) = att'$. Moreover, let F_y be the forward reachable set of y in G'. Then $F_y \subseteq att$. There is an attractor att'_y of G' such that $att'_y \subseteq F_y$. So, we can also choose the mapping m such that $m(s) = s$ for $s \in A \backslash \{att\}$ and $m(att) = att'_y$. Since att' may be different to att'_y, the mapping m may not be uniquely determined. For example, consider the ARBN of the BN in Example 1. Its STG G is as in Figure 2b. We have, $A = \{\{000\}, \{101, 111\}\}$, $G' = G - (111, 101)$. Then, $A' = \{\{000\}, \{111\}\}$. Obviously, there is a mapping m where $m(\{000\}) = \{000\}$ and $m(\{101, 111\}) = \{111\}$.

From Case 1 and Case 2, there exists a mapping $m : A \to A'$ with $m(att) \subseteq att$ for all $att \in A$. Since attractors are pairwise disjoint, $m(att_1) \neq m(att_2)$ for all $att_1, att_2 \in A, att_1 \neq att_2$. Hence, m satisfies (1).

Therefore, we can conclude the proof. □

Corollary 1. *Let $\mathcal{N}$ be a BN and its STG be G. G' is the graph obtained by removing an arc (x, y) from G. Let A and A' be the sets of attractors of G and G', respectively. Then, $|A'| \geq |A|$.*

Proof: By Lemma 3, there is a mapping $m : A \to A'$ with $m(att) \subseteq att$ for all $att \in A$ and $m(att_1) \neq m(att_2)$ for all $att_1, att_2 \in A, att_1 \neq att_2$. Obviously, m is an injection. Hence, $|A'| \geq |A|$. □

Theorem 2. *Let $\mathcal{N}$ be a BN and its STG be G. G' is the graph obtained by removing arcs from G. Let A and A' be the sets of attractors of G and G', respectively. Then, the exists a mapping $m : A \to A'$ with $m(att) \subseteq att$ for all $att \in A$ and $m(att_1) \neq m(att_2)$ for all $att_1, att_2 \in A, att_1 \neq att_2$ (1).*

Proof: We prove this theorem by using induction on p (the number of the removed arcs) and using Lemma 3.

Without loss of generality, we order the removed arcs as $e_1, ..., e_p$, $p \geq 1$. The base case, $p = 1$, clearly holds. Indeed, there is a mapping m satisfying (1) by Lemma 3. Assume that there is a mapping m satisfying (1) for all $p \leq k$.

The inductive case, $p = k + 1$, also holds. $G' = G - \{e_1, ..., e_{k+1}\}$. Let $G'' = G - \{e_1, ..., e_k\}$ and A'' be the set of attractors of G''. By the induction hypothesis, we have a mapping $m_1 : A \to A''$ with $m_1(att) \subseteq att$ for all $att \in A$ and $m_1(att_1) \neq m_1(att_2)$ for all $att_1, att_2 \in A, att_1 \neq att_2$. $G' = G'' - \{e_{k+1}\}$. By Lemma 3, we have a mapping $m_2 : A'' \to A'$ with $m_2(att) \subseteq att$ for all $att \in A''$ and $m_2(att_1) \neq m_2(att_2)$ for all $att_1, att_2 \in A'', att_1 \neq att_2$. Choose $m = m_2 \circ m_1$. Clearly, m satisfies (1).

Therefore, we can conclude the proof. □

Note that these lemmas and theorems do not depend on the updating scheme of the BN. This allows to extend our method for ARBNs presented in this paper to that for other types of BNs.

4 FVS-BASED METHOD

From the relations presented in Section 3, we propose an FVS-based method for finding all attractors (fixed points and cyclic attractors) of an ARBN. This method includes many steps. We first show the general approach of our method. Then, we show each step in detail.

4.1 General approach

The intuitive idea of our method is as follows. Given an ARBN $\mathcal{A}$, let G be the STG of $\mathcal{A}$. We systematically remove arcs from G to obtain a new acyclic STG G'. Let F be the set of fixed points of G'. The calculation of F will be deeply discussed in Subsection 4.2. In G, we then filter out F to get the new set A which one-to-one corresponds to the set of attractors of $\mathcal{A}$. The obtained set is sufficient since starting from a state s in an attractor, we can enumerate this attractor by listing all states reachable from s. We can compactly represent $FR^{\mathcal{A}}(\{s\})$ as a BDD [12] or a finite complete prefix [27]. Note that F also contains the set of fixed points of G (say F_{fix}) and F_{fix} can be easily calculated by using BDD. Specifically, F_{fix} can be represented as a BDD characterized by $\bigwedge_{i=1}^{n}(x_i \leftrightarrow f_i(x))$. Thus, we can remove F_{fix} from F before filtering the set F.

The description of our method is shown in Algorithm 2. Note that, the result of Algorithm 2 is a set of states where each state represents (i.e., belongs to) an attractor of the ARBN. Let see an example as follows. Consider the ARBN of the BN in Example 1. The STG G of this ARBN is given as in Figure 2b. Choose $U = \{x_1, x_2\}, b_1 = 0, b_2 = 1$. The reduced STG G' is given as in Figure 4b. Then, $F = \{000, 010, 111\}$ and $F_{fix} = \{000\}$. After finishing Line 8, we have $F = \{010, 111\}$ and $A = \{000\}$. Suppose that $s = 010$ in the first iteration of the *while* loop. Since 010 reaches 000 in the STG G, 010 is not added to A. In the next iteration, $s = 111$, and it is added to A since 111 does not reach any state in $A \cup F = \{000\}$. Finally, $A = \{000, 111\}$ where 000 represents the attractor $\{000\}$ and 111 represents the attractor $\{111, 101\}$.

Algorithm 2 *Algorithm for finding all attractors of an ARBN*

```
Input: An ARBN $\mathcal{A}$
Output: The set $A$ of all attractors of $\mathcal{A}$
 1: Find an FVS $U = \{x_{i_1}, ..., x_{i_k}\}$ of $\mathcal{A}$
 2: Choose a set $B = \{b_{i_1}, ..., b_{i_k}\}$ of retained values corre-
    sponding to the nodes of $U$
 3: Let $G$ be the STG of $\mathcal{A}$
 4: Let $G'$ be the STG obtained by removing all arcs $(x, x')$
    from $G$ where $\bigvee_{j=1}^{k}(x_{i_j} \leftrightarrow b_{i_j} \wedge x'_{i_j} \leftrightarrow 1 - b_{i_j})$ holds
 5: $F \leftarrow$ the set of fixed points of $G'$
 6: $F_{fix} \leftarrow$ the set of fixed points of $G$
 7: $F \leftarrow F \backslash F_{fix}$
 8: $A \leftarrow F_{fix}$
 9: while $F \neq \emptyset$ do
10:     Remove a state $s$ from $F$
11:     if $s$ does not reach in $G$ any state in $A \cup F$ then
12:         $A \leftarrow A \cup \{s\}$
13:     end if
14: end while
15: return $A$
```

Theorem 3. *Algorithm 2 finds exactly all attractors of an ARBN.*

Proof: By Theorem 1, G' has no cycles. That means F is the set of all attractors of G'.

After finishing Line 8, each attractor of G contains at least one state in $A \cup F$ by Theorem 2 (1). Through the *while* loop, the property (1) is preserved. Indeed, in each iteration, if s reaches a state s' in $A \cup F$ and s is in an attractor att of G, then s' is also in att by the definition of an attractor.

After finishing the *while* loop, each attractor of G contains at least one state in A (a) since $F = \emptyset$. We show that A has no two states s and s' such that s and s' are in the same attractor of G (b). Indeed, if s and s' is in the same attractor of G, then s (s') reaches s' (s). If s is traversed before s', then s can not be added to A. If s' is traversed before s, then s' can not be added to A. Moreover, there is no state s in A such that s is not in any attractors (c). Indeed, if such a state s exists, it must reach at least an attractor, implying that it reaches $A \cup F$. Hence, s can not be added to A. This is a contradiction.

From (a), (b), and (c), we have that A one-to-one corresponds to the set of attractors of G. Therefore, Algorithm 2 finds exactly all attractors of the ARBN. □

In the following subsections, we will present the steps of Algorithm 2 in detail. For each step, we formalize and analyze the problems related to it. Then, we propose methods to efficiently solve these problems.

4.2 Finding fixed points of the reduced STG

First, we consider the problem of calculating the set of fixed points of the reduced STG G'.

Problem 1:
Instance: An ARBN $\mathcal{A} = (V^{\mathcal{A}}, F^{\mathcal{A}})$, an FVS U of $\mathcal{A}$, a retained set B.
Question: How can we efficiently find the set of fixed points of the reduced STG G' as described in the previous sections?

In an ARBN, a state s in G will become a fixed point in G' if and only if the updating of the node x_i does not change s_i for all $x_i \in V^{\mathcal{A}} \backslash U$ and the updating of the node x_i does not change s_i or changes s_i from b_i to $1 - b_i$ for all $x_i \in U$. Obviously, the set F of fixed points of G' can be characterized by a propositional formula. For example, let consider the ARBN of the BN in Example 1. Choose $U = \{x_1, x_2\}$. Then, F can be characterized by the following formula:

$$\begin{aligned}F_{char} = &((x'_1 \leftrightarrow x_1) \vee (x_1 \leftrightarrow b_1 \wedge x'_1 \leftrightarrow 1 - b_1))\\ &\wedge ((x'_2 \leftrightarrow x_2) \vee (x_2 \leftrightarrow b_2 \wedge x'_2 \leftrightarrow 1 - b_2))\\ &\wedge (x'_3 \leftrightarrow x_3)\\ &\wedge (x'_1 \leftrightarrow x_2 \vee x_3) \wedge (x'_2 \leftrightarrow x_1 \wedge \neg x_2) \wedge (x'_3 \leftrightarrow x_1)\end{aligned}$$

where x and x' denote the current state and the next state, respectively. Clearly, x'_1, x'_2, x'_3 can be eliminated from F_{char}. We obtain a new formula of F_{char} whose variables are only x_1, x_2, x_3. We have $F_{char} = ((x_2 \vee x_3 \leftrightarrow x_1) \vee (x_1 \leftrightarrow b_1 \wedge (x_2 \vee x_3 \leftrightarrow 1 - b_1))) \wedge ((x_1 \wedge \neg x_2 \leftrightarrow x_2) \vee (x_2 \leftrightarrow b_2 \wedge (x_1 \wedge \neg x_2 \leftrightarrow 1 - b_2))) \wedge (x_1 \leftrightarrow x_3)$.

We can generalize the formula F_{char} as $F_{char} = \bigwedge_{x_i \in U}((f_i(x) \leftrightarrow x_i) \vee (x_i \leftrightarrow b_i \wedge f_i(x) \leftrightarrow 1 - b_i)) \wedge \bigwedge_{x_i \notin U}(f_i(x) \leftrightarrow x_i)$. Since U is an FVS, the value of $x_i \notin U$ is uniquely determined from the values of the nodes of U. That means we can obtain a formula which is equivalent to F_{char} and contains only the variables of the nodes of U. Thus, we can claim that $|F|$ (the number of fixed points of G') is at most $2^{|U|}$. In real biological networks, the size of the minimum FVS is often much smaller than n. Hence, $|F|$ may be much smaller than the number of possible states of the ARBN (2^n) if we use the minimum FVS as U. Note that $2^{|U|}$ is only an upper bound of $|F|$, and $|F|$ depends on both U and B. Thus, having the minimum FVS may not ensure having the smallest $|F|$ (see Example 2).

Example 2. *Consider the BN in Example 1. The interaction graph of this BN has three FVSs including $\{x_1, x_2\}$, $\{x_2, x_3\}$, and $\{x_1, x_2, x_3\}$. Herein, $\{x_1, x_2\}$ and $\{x_2, x_3\}$ two minimum FVSs. If we choose $U = \{x_1, x_2\}, b_1 = 0, b_2 = 1$, then $|F| = |\{000, 010, 111\}| = 3$. If we choose $U = \{x_1, x_2, x_3\}, b_1 = 0, b_2 = 0, b_3 = 0$, then $|F| = |\{000, 101\}| = 2$.*

The authors of [19] use an enumeration-based approach which exhausts all $2^{|U|}$ possible values of the nodes of U. However, since F_{char} is a propositional formula, we can use some techniques, such as, BDD or SAT (All-SAT). BDD is often inefficient for large networks. In this paper, we therefore use BDD and All-SAT to calculate F for networks with $n \leq 60$ and $n > 60$, respectively.

Obviously, $|F|$ depends on the value of the set B. Consider the ARBN of the BN in Example 1. If we choose $U = \{x_1, x_2\}, b_1 = 0, b_2 = 0$, then $|F| = |\{000, 101\}| = 2$ (see Figure 4a). If we choose $U = \{x_1, x_2\}, b_1 = 0, b_2 = 1$, then $|F| = |\{000, 010, 111\}| = 3$ (see Figure 4b). We expect that $|F|$ is as small as possible since the number of iterations of Algorithm 2 is equal to $|F| - |F_{fix}|$ and $|F_{fix}|$ is fixed with respect to the ARBN. In next, we consider the problem of efficiently choosing the set B to obtain this expectation.

Problem 2:
Instance: An ARBN $\mathcal{A} = (V^{\mathcal{A}}, F^{\mathcal{A}})$, an FVS U of $\mathcal{A}$, a retained set B.
Question: Find a value for the retained set B such that the set of fixed points of the reduced STG G' is minimum.

Let F denote the set of fixed points of the reduced STG G'. Then, F can be represented as a propositional formula F_{char} of n Boolean variables $x_1, ..., x_n$ and m Boolean parameters $b_{i_1}, ..., b_{i_m}$ where $0 \leq m \leq n$. Problem 2 aims to find an assignment $b^*_{i_1}, ..., b^*_{i_m}$ to $b_{i_1}, ..., b_{i_m}$ such that the number satisfying assignments of $F_{char}(b_{i_1}/b^*_{i_1}, ..., b_{i_m}/b^*_{i_m})$ to $x_1, ..., x_n$ (the number of fixed points) is minimum. For example, consider the ARBN of the BN in Example 1. Choose $U = \{x_1, x_2\}$. Then, F can be characterized by the following formula:

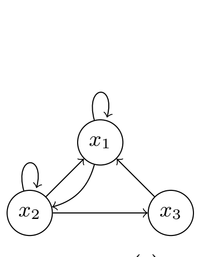
$$\begin{aligned}F_{char} = &(((x_2 \vee x_3) \leftrightarrow x_1) \vee (x_1 \leftrightarrow b_1 \wedge (x_2 \vee x_3) \leftrightarrow 1 - b_1))\\ &\wedge (((x_1 \wedge \neg x_2) \leftrightarrow x_2) \vee (x_2 \leftrightarrow b_2 \wedge (x_1 \wedge \neg x_2) \leftrightarrow 1 - b_2))\\ &\wedge (x_1 \leftrightarrow x_3).\end{aligned}$$

If $(b^*_1, b^*_2) = (0, 0)$, then the number of satisfying assignments of $F_{char}(b_1/b^*_1, b_2/b^*_2)$ is 2, i.e., $|F| = 2$ (see Figure 4a).
If $(b^*_1, b^*_2) = (0, 1)$, then the number of satisfying assignments of $F_{char}(b_1/b^*_1, b_2/b^*_2)$ is 3, i.e., $|F| = 3$ (see Figure 4b).

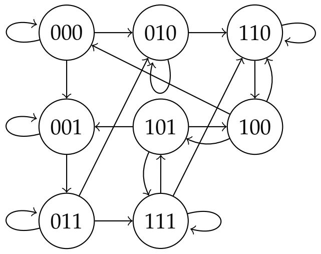
This problem is related to the problem of counting the number of satisfying assignments of a propositional formula which is known as a #P-complete problem. Let $\mathcal{P}$ denote Problem 2. Here, we consider the constructive version of $\mathcal{P}$ (i.e., the output of $\mathcal{P}$ includes the optimal assignment to $b_{i_1}, ..., b_{i_m}$ and the minimum number of satisfying assignments to $x_1, ..., x_n$). $\mathcal{P}$ seems to be an optimization problem whose measure function is a #P function, thus it seems to be in Opt#P [29]. The proof would be very technical and long. However, we can see that it is too hard to solve $\mathcal{P}$. In this paper, we use a heuristic method to solve it.

Let see the characterized formula F_{char} of F. Intuitively, we need to assign the most evaluation of f_{i_k} ($k \in \{1, ..., m\}$) to b_{i_k}. $a \in \{0, 1\}$ is the most evaluation of a function f if the number of satisfying assignments on f's inputs of $f \leftrightarrow a$ is greater than or equal to the number of satisfying assignments on f's inputs of $f \leftrightarrow 1 - a$. Assigning the most evaluation of f_{i_k} to b_{i_k} makes the number of assignments to $x_1, ..., x_n$ of $(x_{i_k} \leftrightarrow b_{i_k} \wedge f_{i_k}(x) \leftrightarrow 1 - b_{i_k})$ decreased. Since the number of assignments to $x_1, ..., x_n$ of $x_{i_k} \leftrightarrow f_{i_k}(x)$ does not depend on b_{i_k}, this rule can reduce the number of assignments to $x_1, ..., x_n$ of $(x_{i_k} \leftrightarrow f_{i_k}(x) \wedge (x_{i_k} \leftrightarrow b_{i_k} \wedge f_{i_k}(x) \leftrightarrow 1 - b_{i_k}))$. For example, the most evaluation of $f_1 = x_2 \vee x_3$ is 1 (from the truth table, 1 (75%) and 0 (25%)), thus we assign 1 to b_1. Similarly, b_2 is assigned to 0. Now, $|F|$ is 2. The number of attractors of the ARBN is 2, we thus obtain the optimal value of $|F|$ since the number of attractor of the ARBN is a lower bound of $|F|$ by Theorem 2. Note that this idea follows a greedy manner, thus the optimality of the solution is not guaranteed (see Example 3).

Example 3. *Consider a BN of three nodes (x_1, x_2, and x_3). Its Boolean functions are given by: $f_1 = (\neg x_1 \vee \neg x_3) \wedge x_2$, $f_2 = \neg x_1 \vee \neg x_2$, $f_3 = \neg x_2$. Figures 5a and 5b denote the interaction graph and the STG of the ARBN counterpart of this BN, respectively. Choose $U = \{x_1, x_2\}$. By the heuristic presented in the previous paragraph, we have $b_1 = 0, b_2 = 1$ and $|F| = |\{010, 110\}| = 2$. However, if we choose $b_1 = 1, b_2 = 1$, we will obtain $|F| = |\{110\}| = 1$.*



(a)



(b)

Fig. 5: An example BN. Its interaction graph (a) and STG of its ARBN counterpart (b).

There is a small problem: How to efficiently calculate the most evaluation of f_{i_k}? Constructing a truth table of f_{i_k} is a direct and simple way. However, when f_{i_k} has many input nodes, this way is inefficient. Since f_{i_k} can be encoded as a BDD, we can use the SATCount function in BDD. The time complexity of SATCount is $\mathcal{O}(|IN(f_{i_k})|)$ [30]. Specifically, we here propose an algorithm for finding an assignment to B. Algorithm 3 shows our algorithm. SATCount(f_i) returns the number of satisfying assignments of f_i. c_{true} and c_{false} denote the numbers of assignments in which the values of f_i are 1 and 0, respectively. In the *for* loop, when $c_{true} = c_{false}$, we randomly assign either 1 or 0 to b_i.

4.3 Preprocessing

Obviously, the number of iterations of Algorithm 2 is equal to $|F|$. Note that all fixed points of the ARBN have been excluded from F (see Line 7 of Algorithm 2). Thus, we here propose a preprocessing which aims at shrinking the set F. We name this preprocessing as Preprocessing SSF (Shrinking the Set F). In each iteration of Preprocessing SSF, we choose randomly a

Algorithm 3 *Algorithm for finding an assignment to B*

```
Input: $B = \{b_{i_1}, ..., b_{i_m}\}$
Output: An assignment $(b^*_{i_1}, ..., b^*_{i_m})$ to $b_{i_1}, ..., b_{i_m}$
 1: for all $i \in \{i_1, ..., i_m\}$ do
 2:     $c_{true} \leftarrow$ SATCount($f_i$)
 3:     $c_{false} \leftarrow 2^{|IN(f_i)|} - c_{true}$
 4:     if $c_{true} > c_{false}$ then
 5:         $b^*_i \leftarrow 1$
 6:     else if $c_{true} < c_{false}$ then
 7:         $b^*_i \leftarrow 0$
 8:     else
 9:         $b^*_i \leftarrow$ randomly either 1 or 0
10:     end if
11: end for
12: return $(b^*_{i_1}, ..., b^*_{i_m})$
```

node x_i to be updated. Then, $F \leftarrow F'$ where F' is the forward image set of F by updating node x_i, i.e., $F' = \bigcup_{s \in F} N(s, i)$ where $N(s, i)$ returns a state s' such that (s, s') is a transition of the STG of the ARBN and $s'_i = f_i(s)$. Note that F' can be easily calculated by using BDD (see [13]). The number of iterations can be empirically obtained. We will discuss this in more detail in the end of this subsection. Hereafter, we show that Preprocessing SSF preserves the correctness of Algorithm 2 by three following properties of ARBNs. We omit the proofs of these properties since they are obvious.

Property 1: Let F' be the forward image set of F by updating node x_i. Then, $|F'| \leq |F|$. Moreover, two states of F may have the same next state by updating node x_i. In this case, we have $|F'| < |F|$.

Property 2: If s is in an attractor of the ARBN, then s' is also in this attractor where $s \rightarrow^* s'$.

Property 3: If s is a state of the ARBN, then $FR(\{s'\}) \subseteq FR(\{s\})$ where $s \rightarrow s'$ and $FR(\{s\})$ is the set of states reachable from $\{s\}$.

Property 1 guarantees that the cardinilaty of F does not increase through these iterations even may decrease. Property 2 guarantees that all cyclic attractors of the ARBN still appear in the set F, thus all attractors of the ARBN still appear in $A \cup F$. Property 3 justifies the usefulness of Preprocessing SSF for the next steps since the forward reachable set of each state in F may be shrunk. Note that after finishing the iterations, F may contain some fixed points of the ARBN. Thus, we need to again exclude the fixed points of the ARBN, i.e., to again perform Line 7 of Algorithm 2. Now, all attractors of the ARBN still appear in $A \cup F$.

To illustrate Preprocessing SSF, we continue with the ARBN of the BN in Example 1. Assume that $U = \{x_1, x_2\}, b_1 = 0, b_2 = 1$. Then, $F = \{000, 010, 111\}$ (see Figure 4b). After finishing Lines 7 and 8 of Algorithm 2, we have $F = \{010, 111\}$ and $A = \{000\}$. Assume that the number of iterations of Preprocessing SSF is 2. Let $I = \langle x_{i_1}, x_{i_2} \rangle$ denote the sequence of the updated nodes. If $I = \langle x_1, x_2 \rangle$, then $F = \{010, 111\} \rightarrow F = \{110, 111\} \rightarrow F = \{110, 101\}$ (see Figure 2b). Obviously, the cardinilaty of F does not increase and all attractors ($\{000\}$ and $\{101, 111\}$) still appear in $A \cup F$ through these iterations. We have $FR(\{110\}) = \{110, 100, 000, 101, 111\}$ and $FR(\{010\}) = \{010, 110, 100, 000, 101, 111\}$. Property 3 holds since $FR(\{110\}) \subseteq FR(\{010\})$. If we choose $I = \langle x_1, x_3 \rangle$, then $F = \{010, 111\} \rightarrow F = \{110, 111\} \rightarrow F = \{111\}$ (see Figure 2b). In this case, the cardinilaty of F even decreases. Moreover, if we choose $I = \langle x_2, x_3 \rangle$, then $F = \{010, 111\} \rightarrow F = \{000, 111\} \rightarrow F = \{000, 111\}$ (see Figure 2b). The cardinilaty of F is unchanged. However, after excluding the fixed points again, we have $F = \{111\}$, i.e., the cardinilaty of F decreases.

Finally, we discuss how to set the number of iterations of Preprocessing SSF (say I_{max}). Obviously, we expect that I_{max} is good enough, i.e., I_{max} is not too large and the cardinilaty of F after Preprocessing SSF is small enough. Preprocessing SSF is optimal when the cardinilaty of F after Preprocessing SSF is equal to the number of cyclic attractors. We set I_{max} based on the following intuitions. First, if n increases (decreases) then I_{max} should increase (decrease). Second, if $|F|$ increases (decreases) then I_{max} should increase (decrease). Third, if F_{fix} increases (decreases) then I_{max} should decreases (increases). Last, I_{max} however should not exceed a threshold. Combining with running some sample networks, we set I_{max} as $I_{max} = min(2 \times n^{1.5} \times |F|/(1+|F_{fix}|), 5000)$. We use $1+|F_{fix}|$ to deal with the case $|F_{fix}| = 0$.

4.4 Reachability analysis

Reachability is a central problem in systems science. It is also the key task in our method. In theory, the reachability in ARBNs is very difficult. Indeed, the PSPACE-completeness of the reachablity in ARBNs has been proved in [31]. However, in practice, there are various methods for checking the reachability in ARBNs. Since the set of states reachable from the starting state may be very large, the explicitly depth-first search and breadth-first search manners are inefficient. We need a more efficient approach. The use of BDD on a breadth-first search-based method is a better solution. However, it still meets the inherent problems of BDD [32] (e.g., extremely long computational time, out of memory). A SAT-based bounded model checking [32] is an efficient approach. However, it is incomplete unless we use a completeness threshold which is usually very hard to compute even for the case of CRBNs [14]. Note that ARBNs have high concurrency [31]. Thus, we can use an unfolding-based approach which is known to be an efficient approach exploiting the concurrency of systems [33]. Recently, some efficient approximation methods (e.g., Pint [34], ASPReach [35]) have been proposed for checking reachability in ARBNs. The authors report that these methods can handle large-scale networks. However, they are of course incomplete. Moreover, in their experiments, the start and target only cover a small set of nodes. In this paper, we aim at proposing an exact and efficient method for finding ARBN attractors. Therefore, we focus on unfolding-based methods for checking the reachability in ARBNs. Considering these approximation methods to apply in our method is one of our future work.

The method for encoding an ARBN as a 1-safe PN has been proposed [27]. Thus, we can apply the algorithms [33] for checking the reachability in 1-safe PNs to that in ARBNs. Hereafter, we show how Line 11 of Algorithm 2 is performed. Obviously, we can perform it by calling $UnfReach(\mathcal{A}, s, A \cup F)$. The description of the function $UnfReach$ is shown in Algorithm 4. However, there are two problems needed to be considered. First, in the case the reachability holds, we do not need to build the whole finite complete prefix $\mathcal{P}_{\mathcal{U}}$. Second, M_F is a set of markings. It is too waste when calling Mole $|M_F|$ times since the standard function of Mole is to check whether a given 1-safe PN reaches a desirable marking.

To deal with these problems, we here propose a new method (called $OnTheFlyMole$) based on an on-the-fly manner. Note that the authors of [27] have also used Petri net unfoldings for the reachability analysis of ARBNs. However, they have not shown in detail their method. Our method may be similar to the method by [27]. We first add new transitions to $\mathcal{P}$. A new transition corresponds to a marking in M_F. For each transition $t_m (m \in M_F)$, we add new arcs from all places in m to t_m. This means that when reaching the marking m, t_m is enabled. Then, in the process of building the finite complete prefix $\mathcal{P}_{\mathcal{U}}$, whenever at least one new transition is enabled (i.e., this transition will appear in $\mathcal{P}_{\mathcal{U}}$) we terminate the process and return true (i.e., reachable). When finishing the process, we return false (i.e., unreachable). Mole also supports an on-the-fly manner. However, it only deals with the case of one transition. Therefore, we have adjusted a little the source code of Mole to implement our method.

Back to the *UnfReach* function, its special case is when $F = \emptyset$. For this case, we simply return false (i.e., unreachable). When $F \neq \emptyset$, we set the corresponding marking of s in $\mathcal{P}$ (denoted by $[[s]]_{\mathcal{P}}$) as the initial marking of $\mathcal{P}$ and then simply call $OnTheFlyMole(\mathcal{P}, M_F)$ where M_F is the set of markings corresponding to the states in F.

Algorithm 4 *UnfReach*

```
Input: An ARBN $\mathcal{A}$, a state $s$, a set $F$ of states
Output: Whether $s$ reaches $F$ on the STG of $\mathcal{A}$?
 1: $F \leftarrow PreprocessingBCN(s, F)$
 2: if $F = \emptyset$ then
 3:     return false
 4: else
 5:     $\mathcal{P} \leftarrow$ the encoded 1-safe PN of $\mathcal{A}$
 6:     $M_0 \leftarrow [[s]]_{\mathcal{P}}$
 7:     Set $M_0$ as the initial marking of $\mathcal{P}$
 8:     $M_F \leftarrow \{[[x]]_{\mathcal{P}} | x \in F\}$
 9:     return $OnTheFlyMole(\mathcal{P}, M_F)$
10: end if
```

The size of a built prefix of the encoded 1-safe PN may be very large. In this case, the computation of finite prefixes may be intractable. To mitigate this issue, we first use a preprocessing step in the *UnfReach* function. This preprocessing aims at solving the reachability in an ARBN without building its prefixes or at least reducing the number of target states (i.e., the states of F). The optimal case is when $F = \emptyset$ implying that s does not reach F in the STG of $\mathcal{A}$. Hereafter, we only show the description of this preprocessing, its usefulness will be discussed in Subsection 5.1.

We first define some types of nodes in a BN as follows. A node $x_i, i \in \{1, ..., n\}$ is called a *zero-constant* or *one-constant* node if $f_i \wedge \neg x_i = 0$ or $f_i \vee \neg x_i = 1$, respectively. In the biological context, a constant node will retain its value once it is set to a specific value (0 or 1). For example, if $f_i = 0$ or $f_i = x_i \wedge ...$, then x_i is called a zero-constant node. In the other hand, if $f_i = 1$ or $f_i = x_i \vee ...$, then x_i is called a one-constant node. The preprocessing is based on the properties of zero-constant and one-constant nodes. We name this preprocessing as Preprocessing BCN (Based on Constant Nodes). Let V^0 and V^1 be the sets of zero-constant nodes and one-constant nodes of $\mathcal{A}$, respectively. For each node $x_i \in V^0$, if $s_i = 0$ we can remove from F the state s' such that $s'_i = 1$. For each node $x_i \in V^1$, if $s_i = 1$ we can remove from F the state s' such that $s'_i = 0$. Now, we obtain a new set F' satisfying s reaches F in the STG of $\mathcal{A}$ if and only if s reaches F' in the STG of $\mathcal{A}$.

5 Experiments

We have implemented the proposed method for finding ARBN attractors in a JAVA tool called FVS-ARBN. FVS-ARBN uses the JDD library [36] for BDD manipulation and Z3 [37] as the SAT solver. An executable file of FVS-ARBN and some examples of real biological networks are available at: https://sites.google.com/site/trinhgiangjaist/. To evaluate the efficiency of Preprocessing SSF, we conduct experiments to compare the performance of two variants of our method. The first variant (say $\mathcal{M}_1$) does not use Preprocessing SSF while the second variant (say $\mathcal{M}_2$) uses Preprocessing SSF. We also compare the performance among $\mathcal{M}_1$, $\mathcal{M}_2$, genYsis [13], and CABEAN [20]. We choose genYsis and CABEAN since they are exact and famous tools for finding attractors of ARBNs.

All the experiments were run on a virtual machine whose environment is CPU: Intel(R) Xeon(R) Silver 4116 4x2.10GHz, Memory: 24 GB, CentOS 7 64 bit. We used two sets of Boolean networks. The first set includes BN models of real biological networks gotten from the literature. The second set includes random BNs generated with Bool Net R package [38]. Note that FVS-ARBN uses BDD in Preprocessing SSF and both genYsis and CABEAN are BDD-based methods. Since high memory consuming is an inherent problem of BDD, memory needs to be considered. In our experiments, we set the heap size to 16 GB. With this heap size, all these algorithms never met the OutOfMemory error before exceeding the time limit in all networks.

5.1 Results on real biological networks

We applied these algorithms to BNs of 32 real biological networks whose sizes range from 19 to 101. A BN can have some input nodes x_i which do not change their values through the evolution of the BN (i.e., $f_i = x_i$). We here consider the networks where the value of an input node is not fixed to either 0 or 1. CABEAN requires to use a network reduction technique which removes all the *leaf nodes* [11] of a BN. This reduction technique conserves the attractors of ARBNs [39]. To ensure the fairness of the experiments, we also used this reduction technique for both FVS-ARBN and genYsis. Lastly, the time limit for each network was set to 10 hours since the running time may be very long for large-scale networks.

The results are shown in Table 1. Columns 1, 2, 3, and 4 denote the name of the network, the number of nodes (n), the size of the FVS ($|U|$), and the number of attractors ($|A|$), respectively. Column "$|F|$" denotes the size of the filtering set F before using Preprocessing SSF while Column "$|F_1|$" denotes that after using Preprocessing SSF. Column "time" denotes the running time in seconds, respectively. "-" stands for the case of not obtaining the result within the time limit. We observed that in some BNs, CABEAN terminated before exceeding the time limit and the Segmentation fault error was printed. We guess that in the computation of attractors CABEAN will terminate when meeting a criterion (e.g., the size of the computed attractor exceeds a threshold). Anyway, CABEAN did not finish the computation of attractors in this case. For these BNs, we report the time when CABEAN terminated and use "*" to indicate the case. From these results, we obtain some remarks as follows.

First, the size of the FVS obtained by Algorithm 1 is much smaller than the network size in all the networks (especially in, e.g., the HGF_Signaling_in_Keratinocytes network, the PC12CellDifferentiation network, the T_Cell_Receptor_Signaling network). This confirms that Algorithm 1 is good enough for finding a minimum or near minimum FVS.

Second, $|F_1|$ is smaller than $|F|$ in 29/32 networks. Moreover, $\mathcal{M}_2$ outperforms $\mathcal{M}_1$ in most networks, especially in, e.g., the FA_BRCA_pathway network, the Bcell network, the MAPK network. Note that in some networks (e.g., the Differentiation_of_T_lymphocyte network, the YeastApoptosis network, the IL_6_Signalling network), $\mathcal{M}_2$ is much slower than $\mathcal{M}_1$. This is apparent since we must take time for Preprocessing SSF. However, the difference between running time of $\mathcal{M}_2$ and $\mathcal{M}_1$ is insignificant or the running time of $\mathcal{M}_2$ is reasonable ($<$ 11 mins) in these networks. These observations are evidence for the usefulness of Preprocessing SSF.

Third, $\mathcal{M}_2$ outperforms genYsis in most networks. In 8/32 networks, genYsis failed to obtain the result within the time limit while $\mathcal{M}_2$ succeeded. In 1/32 networks (the YeastApoptosis network), $\mathcal{M}_2$ is slower than genYsis. However, the difference between running time of $\mathcal{M}_2$ and genYsis is insignificant. In most of 23/32 remaining networks, $\mathcal{M}_2$ is much faster than genYsis. Especially, the difference between running time of $\mathcal{M}_2$ and genYsis is very large in some of these networks, e.g., the Bcell network (22.59 and 8702.80), the HumanMyelomaCells network (47.00 and 12983.39), the TcellLGL network (55.56 and 21198.63). Moreover, $\mathcal{M}_1$ even outperforms genYsis in some networks (e.g., the Colitis_associated_colon_cancer network, the Differentiation_of_T_lymphocytes network, the IL_6_Signalling network). These observations show the effectiveness of the FVS-based method as compared to genYsis.

Fourth, $\mathcal{M}_2$ also outperforms CABEAN in most networks. In 10/32 networks, CABEAN failed to finish the computation of attractors while $\mathcal{M}_2$ succeeded within the time limit. Even the running time of CABEAN before terminating is greater (e.g., the T_Cell_Receptor_Signaling network) or much greater (e.g., the InflammatoryBowelDisease network, the TLGLSurvival network, the Colitis_associated_colon_cancer network) than the running time of $\mathcal{M}_2$. In 9/32 networks, $\mathcal{M}_2$ is slower than CABEAN. However, the difference between running time of $\mathcal{M}_2$ and genYsis is insignificant (e.g., the AuroraKinaseA network, the GuardCellAbscisicAcidSignaling network) or the running time of $\mathcal{M}_2$ is reasonable (e.g., the Differentiation_of_T_lymphocytes network, the YeastApoptosis network). In most of 13/32 remaining networks, $\mathcal{M}_2$ is much faster than CABEAN. Especially, the difference between running time of $\mathcal{M}_2$ and CABEAN is very large in some of these networks, e.g., the TcellLGL network (55.56 and 916.23), the Drosophila network (4.88 and 1984.40). Moreover, $\mathcal{M}_1$ even outperforms CABEAN in some networks (e.g., the Colitis_associated_colon_cancer network, the Differentiation_of_T_lymphocytes network, the IL_6_Signalling network, the T_Cell_Receptor_Signaling network). These observations show the effectiveness of the FVS-based method as compared to CABEAN.

Last, $\mathcal{M}_1$ and $\mathcal{M}_2$ even can handle large networks in terms of attractor detection in ARBNs without using any network reduction technique. We here report the running time of $\mathcal{M}_1$ and $\mathcal{M}_2$ for some large networks without using any network reduction technique. For the IL_6_Signalling network ($n = 86$), the running time of $\mathcal{M}_1$ and $\mathcal{M}_2$ are 1323.38 seconds and 368.56 seconds, respectively. For the T_Cell_Receptor_Signaling network ($n = 101$), the running time of $\mathcal{M}_1$ and $\mathcal{M}_2$ are 583.71 seconds and 1463.30 seconds, respectively. Note that genYsis failed to obtain the result within the time limit in all the two networks. The analysis for these networks was usually performed by using their reduced versions (e.g., removed leaf nodes or fixed input nodes) because of the performance limitations of the existing tools. Therefore, the advanced computation capability of our method can enable biologists to conduct more accurate analysis on large networks.

In this end of this subsection, we will discuss the usefulness of Preprocessing BCN presented in Subsection 4.4 as well as the impact of the picked FVS to the performance of our method (i.e., $\mathcal{M}_2$).

By applying $\mathcal{M}_2$ without using Preprocessing BCN on some real networks of the benchmark, we observed that Preprocessing BCN can accelerate the running time of our method. In the remy_tumorigenesis network, the speedup by using Preprocessing BCN is $9.13/3.35 = 2.73$. Especially, in the IL_6_Signalling and T_Cell_Receptor_Signaling networks, $\mathcal{M}_2$ without using Preprocessing BCN did not find all the attractors within 10 hours while $\mathcal{M}_2$ with using Preprocessing BCN found all the attractors in 297.51 seconds and 5.27 seconds, respectively (see Table 1).

By applying $\mathcal{M}_2$ with randomly choosing an FVS on some real networks of the benchmark, we observed that the picked FVS may largely impact the performance of our method. In the IL_6_Signalling network, we obtained the result as $|U| = 25$ and the running time is 3337.85 seconds. The result by using Algorithm 1 for choosing an FVS is $|U| = 21$ and the running time is 297.51 seconds (see Table 1). Especially, in the Butanol-Production network, we obtained the result as $|U| = 30$ and $\mathcal{M}_2$ did not find all attractors within 10 hours while the result by using Algorithm 1 for choosing an FVS is $|U| = 18$ and the running time is 324.22 seconds (see Table 1). We have shown in Subsection 4.2 that having the minimum FVS may not ensure having the best performance. However, using a minimum or nearly minimum FVS is still a good strategy to obtain good performance at least in our benchmarks.

5.2 Results on randomly generated networks

We randomly generated a set of N-K BNs [5] with network size n in the set {50, 55, 60, 65, 70, 75, 80, 85, 90, 95, 100, 105, 110} and $K = 2$ (i.e., each node has exactly $K = 2$ input nodes). For each network size, 20 instances were generated. In total, we have 260 random BNs.

We then applied $\mathcal{M}_2$, genYsis, and CABEAN to the 260 random BNs and recorded the number of failures (i.e., failed to obtain the result within 30 minutes). Since the usefulness of Preprocessing SSF has been justified in the previous subsection, we did not apply $\mathcal{M}_1$ to these BNs. Note that we also used the network reduction technique as in Subsection 5.1. The results are shown in Figure 6. As we can see, the number of failures of genYsis or CABEAN rapidly approaches 20. On the other hand, $\mathcal{M}_2$ can even handle 30 or 55 percent of networks for $n = 105$ or $n = 110$, respectively. Moreover, in each network size, the number of failures of genYsis or CABEAN is always larger than the number of failures of $\mathcal{M}_2$. These observations show that $\mathcal{M}_2$ is more scalable than both genYsis and CABEAN in terms of N-K BNs.

For $n = 110$, $\mathcal{M}_2$ failed to obtain the result within 30 minutes in 45 percent of networks. Note that these BNs have been reduced by using the network reduction technique based on leaf nodes. In the biological context, 110-node networks are not very large since a comprehensive analysis of gene regulatory networks often requires formal models including hundreds or even thousands of elements [20]. Thus, $\mathcal{M}_2$ (as also other existing methods) is generally incapable of handling very large BNs (e.g., 500-node or 1000-node BNs). Improving $\mathcal{M}_2$ to handle such BNs is one of our future work.

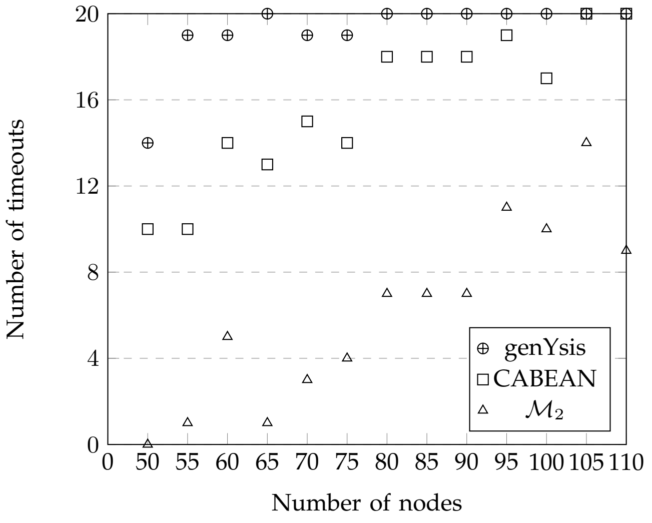


Fig. 6: The results on randomly generated networks.

6 Discussion and future work

We have presented and formally proved the relations between FVSs and dynamics of BNs. From these relations, we proposed an FVS-based method for detecting attractors in ARBNs. Our approach relies on the principle of removing arcs in the STG to get a candidate set and the reachability property to filter the candidate set. The obtained set one-to-one corresponds to the set of all attractors of the ARBN. The correctness of our method has been formally proved. We have also proposed Preprocessing SSF to reduce the computational burden while preserving the correctness of our method. Then, we have proposed an unfolding-based and on-the-fly method for checking reachability property. In principle, our method works well on large networks having relatively small FVSs and not too large attractors. Fortunately, these characteristics are often found in real biological networks [3], [19].

We have implemented our method and conducted experiments on real biological networks and randomly generated N-K networks. The obtained results confirm the usefulness of Preprocessing SSF and are very promising since our method can handle large networks whose sizes are up to 101 without using any network reduction technique. The experimental results also show the effectiveness of our method as compared to the two existing methods, genYsis and CABEAN. Note that the main advantage of our method is to reduce the attractor detection in ARBNs to the reachability problem. This opens a chance to efficiently solve the attractor detection in ARBNs by applying the advents in reachability research which can handle very large asynchronous networks [34].

There are some possible ways to improve our method. The first is to reduce the number of fixed points of the reduced STG. We can use an exact method for finding minimum FVSs or a more efficient heuristic for setting the retained set B. The second is to propose an efficient heuristic for variable ordering since Preprocessing SSF uses BDD. A good variable ordering can reduce significantly the time of Preprocessing SSF. Last is

TABLE 1: Results on real biological networks.

				$\mathcal{M}_1$		$\mathcal{M}_2$			genYsis	CABEAN
name	n	$\lvert U\rvert$	$\lvert A\rvert$	$\lvert F\rvert$	time	$\lvert F\rvert$	$\lvert F_1\rvert$	time	time	time
OxidativeStressPathway [40]	19	4	2	4	0.71	4	1	0.20	1.4	0.73
AuroraKinaseA [40]	23	8	32	32	0.61	32	16	0.73	2.75	0.49
dahlhaus_neuroplastoma [41]	23	8	32	32	0.64	32	16	0.65	2.73	0.54
FA_BRCA_pathway [40]	28	11	1	13	-	13	2	0.73	10.02	4.19
Septation_Initiation_Network [40]	31	10	640	320	1.43	320	0	1.12	51.92	1.22
TumourCell [40]	32	13	9	117	1.75	117	0	0.60	2.33	0.51
Bordetella_bronchiseptica [40]	33	9	3	18	61.85	18	0	0.55	2.03	1.73
Lymphoid_myeloid_cell_specification [40]	33	8	21	35	1.75	35	0	0.59	21.69	3.51
CholesterolRegulatoryPathway [40]	34	3	4	2	0.44	2	0	0.25	4.85	0.49
remy_tumorigenesis [42]	35	16	25	892	72.08	892	5	3.35	15.91	2.49*
TCellSignaling [40]	40	6	8	5	0.46	5	1	0.16	0.22	0.04
ApoptosisNetwork [40]	41	7	8	8	6.08	12	8	7.27	581.07	6.62*
Treatment_of_Castration_Resistant [40]	42	14	16384	0	0.49	0	0	0.13	18.18	0.73
GuardCellAbscisicAcidSignaling [40]	44	8	28	20	0.61	32	15	1.33	7.90	0.83
InflammatoryBowelDisease [40]	47	22	1	960	-	960	1	2.47	-	12.77*
Stomatal_Opening_Model [40]	49	13	48	390	9.48	243	14	10.99	31.22	2.38
Differentiation_of_T_lymphocytes [40]	50	18	2050	5581	80.37	5581	0	627.76	-	89.75
Senescence [40]	51	12	17	84	13.69	84	2	9.93	18.05	3.00
Drosophila [43]	52	14	128	84	1.46	84	0	4.88	-	1984.40
MAPK [40]	53	10	18	102	-	226	6	8.15	-	5.54*
B_bronchiseptica_T_retortaeformis [40]	53	15	30	298	7794.31	298	0	15.61	3556.85	440.16
TcellLGL [40]	60	23	142	11156	-	11156	108	55.56	21198.63	916.23
TLGLSurvival [40]	61	25	318	18276	-	18276	260	174.66	-	1479.23*
PC12CellDifferentiation [40]	62	3	3	0	0.39	0	0	0.20	5.01	0.59
ButanolProduction [40]	66	18	8192	12416	-	12416	6144	324.22	-	24.80*
HumanMyelomaCells [40]	67	14	83	558	1305.75	558	0	47.00	12983.39	0.02*
HGF_Signaling_in_Keratinocytes [40]	68	10	72	256	5.3	256	0	3.79	1200.04	8.75
Colitis_associated_colon_cancer [40]	70	13	10	84	516.06	100	14	391.05	-	12614.96*
Bcell [44]	72	19	72	934	12912.62	934	69	22.59	8702.80	29.84
YeastApoptosis [40]	73	17	8448	4352	3.08	4352	4352	75.32	45.85	1.16
IL_6_Signalling [40]	86	21	32768	20480	104.14	20480	4096	297.51	-	11.71*
T_Cell_Receptor_Signaling [40]	101	10	128	72	2.89	72	24	5.27	3596.65	6.35*

to use a more efficient method for checking the reachability in ARBNs. We can consider some other techniques in terms of Petri net unfoldings, such as, contextual Petri nets [45], merged processes [46]. Moreover, we can also use some static analyzers (e.g., Pint [34], ASPReach [35]) in a preprocessing step.

Our method uses Mole and some other tools as subroutines. Since the computational complexities of such tools are unclear, it is difficult to analyze the computational complexity of the whole algorithm. We leave the analysis of theoretical and/or practical computational complexity as future work. Moreover, random order asynchronous Boolean networks (ROABNs) [18] are also a popular type of asynchronous BNs and still get much attention from research communities [4], [47]. Therefore, it is worthy to study attractors of ROABNs. In the future, we intend to extend our method for ARBNs to that for ROABNs. It is potentially possible since the relations presented in this paper still hold for ROABNs.

Acknowledgments

TA was partially supported by Grant-in-Aid #18H04113 from JSPS, Japan.

References

[1] S. A. Kauffman, *The origins of order: Self-organization and selection in evolution*. OUP USA, 1993.

[2] C. Gershenson, "Introduction to random boolean networks," *arXiv preprint nlin/0408006*, 2004.

[3] ——, "Classification of random boolean networks," *arXiv preprint cs/0208001*, 2002.

[4] A. Saadatpour, I. Albert, and R. Albert, "Attractor analysis of asynchronous boolean models of signal transduction networks," *Journal of theoretical biology*, vol. 266, no. 4, pp. 641–656, 2010.

[5] S. A. Kauffman, "The origins of order: Self-organization and selection in evolution," in *Spin glasses and biology*. World Scientific, 1992, pp. 61–100.

[6] S. Huang, "Genomics, complexity and drug discovery: insights from boolean network models of cellular regulation," *Pharmacogenomics*, vol. 2, no. 3, pp. 203–222, 2001.

[7] S. Yamanaka, "Elite and stochastic models for induced pluripotent stem cell generation," *Nature*, vol. 460, no. 7251, p. 49, 2009.

[8] K. Tun, M. Menghini, L. D'Andrea, P. Dhar, H. Tanaka, and A. Giuliani, "Why so few drug targets: a mathematical explanation?" *Current computer-aided drug design*, vol. 7, no. 3, pp. 206–213, 2011.

[9] R. Somogyi and L. D. Greller, "The dynamics of molecular networks: applications to therapeutic discovery," *Drug Discovery Today*, vol. 6, no. 24, pp. 1267–1277, 2001.

[10] S.-Q. Zhang, M. Hayashida, T. Akutsu, W.-K. Ching, and M. K. Ng, "Algorithms for finding small attractors in boolean networks," *EURASIP Journal on Bioinformatics and Systems Biology*, vol. 2007, pp. 4–4, 2007.

[11] Q. Yuan, H. Qu, J. Pang, and A. Mizera, "Improving bdd-based attractor detection for synchronous boolean networks," *Science China Information Sciences*, vol. 59, no. 8, p. 080101, 2016.

[12] A. Garg, I. Xenarios, L. Mendoza, and G. DeMicheli, "An efficient method for dynamic analysis of gene regulatory networks and in silico gene perturbation experiments," in *Annual International Conference on Research in Computational Molecular Biology*. Springer, 2007, pp. 62–76.

[13] A. Garg, A. Di Cara, I. Xenarios, L. Mendoza, and G. De Micheli, "Synchronous versus asynchronous modeling of gene regulatory networks," *Bioinformatics*, vol. 24, no. 17, pp. 1917–1925, 2008.

[14] E. Dubrova and M. Teslenko, "A sat-based algorithm for finding attractors in synchronous boolean networks," *IEEE/ACM transactions on computational biology and bioinformatics*, vol. 8, no. 5, pp. 1393–1399, 2011.

[15] Y. Zhao, J. Kim, and M. Filippone, "Aggregation algorithm towards large-scale boolean network analysis," *IEEE Transactions on Automatic Control*, vol. 58, no. 8, pp. 1976–1985, 2013.

[16] W. Guo, G. Yang, W. Wu, L. He, and M. Sun, "A parallel attractor finding algorithm based on boolean satisfiability for genetic regulatory networks," *PloS one*, vol. 9, no. 4, p. e94258, 2014.

[17] R. Thomas, "Regulatory networks seen as asynchronous automata: a logical description." *Journal of theoretical biology*, vol. 153, pp. 1–23, 1991.

[18] I. Harvey and T. Bossomaier, "Time out of joint: Attractors in asynchronous random boolean networks," in *Proceedings of the Fourth European Conference on Artificial Life*. MIT Press, Cambridge, 1997, pp. 67–75.

[19] T. Skodawessely and K. Klemm, "Finding attractors in asynchronous boolean dynamics," *Advances in Complex Systems*, vol. 14, no. 03, pp. 439–449, 2011.

[20] A. Mizera, J. Pang, H. Qu, and Q. Yuan, "Taming asynchrony for attractor detection in large boolean networks," *IEEE/ACM transactions on computational biology and bioinformatics*, vol. 16, no. 1, pp. 31–42, 2018.

[21] S. A. Kauffman, "Metabolic stability and epigenesis in randomly constructed genetic nets," *Journal of theoretical biology*, vol. 22, no. 3, pp. 437–467, 1969.

[22] D. S. Johnson and M. R. Garey, *Computers and intractability: A guide to the theory of NP-completeness*. WH Freeman, 1979.

[23] G. Even, J. S. Naor, B. Schieber, and M. Sudan, "Approximating minimum feedback sets and multicuts in directed graphs," *Algorithmica*, vol. 20, no. 2, pp. 151–174, 1998.

[24] R. Tarjan, "Depth-first search and linear graph algorithms," *SIAM journal on computing*, vol. 1, no. 2, pp. 146–160, 1972.

[25] T. Murata, "Petri nets: Properties, analysis and applications," *Proceedings of the IEEE*, vol. 77, no. 4, pp. 541–580, 1989.

[26] J. Esparza and K. Heljanko, *Unfoldings: a partial-order approach to model checking*. Springer Science & Business Media, 2008.

[27] T. Chatain, S. Haar, L. Jezequel, L. Paulevé, and S. Schwoon, "Characterization of reachable attractors using petri net unfoldings," in *International Conference on Computational Methods in Systems Biology*. Springer, 2014, pp. 129–142.

[28] S. Schwoon and S. Romer, "Mole—a petri net unfolder," http://www.lsv.fr/~schwoon/tools/mole/, 2016.

[29] T. Yamakami, "Quantum optimization problems," 2002.

[30] R. E. Bryant, "Graph-based algorithms for boolean function manipulation," *Computers, IEEE Transactions on*, vol. 100, no. 8, pp. 677–691, 1986.

[31] T. Chatain, S. Haar, J. Kolčák, L. Paulevé, and A. Thakkar, "Concurrency in boolean networks," *Natural Computing*, pp. 1–19, 2019.

[32] A. Biere, A. Cimatti, E. M. Clarke, M. Fujita, and Y. Zhu, "Symbolic model checking using sat procedures instead of bdds," in *Proceedings 1999 Design Automation Conference (Cat. No. 99CH36361)*. IEEE, 1999, pp. 317–320.

[33] J. Esparza and C. Schröter, "Unfolding based algorithms for the reachability problem," *Fundamenta Informaticae*, vol. 47, no. 3-4, pp. 231–245, 2001.

[34] L. Paulevé, "Pint: A static analyzer for transient dynamics of qualitative networks with ipython interface," in *International Conference on Computational Methods in Systems Biology*. Springer, 2017, pp. 309–316.

[35] X. Chai, T. Ribeiro, M. Magnin, O. Roux, and K. Inoue, "Static analysis and stochastic search for reachability problem," 2018.

[36] A. Vahidi, "Jdd: a pure java bdd and z-bdd library," https://bitbucket.org/vahidi/jdd, 2003.

[37] L. De Moura and N. Bjørner, "Z3: An efficient smt solver," in *International conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2008, pp. 337–340.

[38] M. Hopfensitz, C. Müssel, M. Maucher, and H. A. Kestler, "Attractors in boolean networks: a tutorial," *Computational Statistics*, vol. 28, no. 1, pp. 19–36, 2013.

[39] A. Naldi, P. T. Monteiro, and C. Chaouiya, "Efficient handling of large signalling-regulatory networks by focusing on their core control," in *International Conference on Computational Methods in Systems Biology*. Springer, 2012, pp. 288–306.

[40] "The Cell Collective," https://cellcollective.org/.

[41] M. Dahlhaus, A. Burkovski, F. Hertwig, C. Mussel, R. Volland, M. Fischer, K.-M. Debatin, H. A. Kestler, and C. Beltinger, "Boolean modeling identifies greatwall/mastl as an important regulator in the aurka network of neuroblastoma," *Cancer letters*, vol. 371, no. 1, pp. 79–89, 2016.

[42] E. Remy, S. Rebouissou, C. Chaouiya, A. Zinovyev, F. Radvanyi, and L. Calzone, "A modeling approach to explain mutually exclusive and co-occurring genetic alterations in bladder tumorigenesis," *Cancer research*, vol. 75, no. 19, pp. 4042–4052, 2015.

[43] M. R. Vega, "Analyzing toys models of arabidopsis and drosphila using z3 smt-lib," in *Independent Component Analyses, Compressive Sampling, Wavelets, Neural Net, Biosystems, and Nanoengineering XII*, vol. 9118. International Society for Optics and Photonics, 2014, p. 911813.

[44] P. Dutta, L. Ma, Y. Ali, P. M. Sloot, and J. Zheng, "Boolean network modeling of β-cell apoptosis and insulin resistance in type 2 diabetes mellitus," *BMC systems biology*, vol. 13, no. 2, p. 36, 2019.

[45] P. Baldan, A. Bruni, A. Corradini, B. König, C. Rodríguez, and S. Schwoon, "Efficient unfolding of contextual petri nets," *Theoretical Computer Science*, vol. 449, pp. 2–22, 2012.

[46] C. Rodríguez, S. Schwoon, and V. Khomenko, "Contextual merged processes," in *International Conference on Applications and Theory of Petri Nets and Concurrency*. Springer, 2013, pp. 29–48.

[47] M. Ishii, J. Gores, and C. Teuscher, "On the sparse percolation of damage in finite non-synchronous random boolean networks," *Physica D: Nonlinear Phenomena*, 2019.

Trinh Van Giang received the B.S. and M.S. degrees in Computer Science from Ho Chi Minh City University of Technology in 2014 and 2017, respectively. He is currently a doctoral student at School of Information Science, Japan Advanced Institute of Science and Technology. His research interests include computational methods for analysis and control of complex dynamical systems.

Tatsuya Akutsu received the B.E. and M.E. degrees in aeronautics and the D.E. degree in information engineering from the University of Tokyo, 1984, 1986, and 1989, respectively. He has been a professor in the Bioinformatics Center, Institute for Chemical Research, Kyoto University since 2001. discrete algorithms. His research interests include bioinformatics and discrete algorithms. He is a senior member of the IEEE.

Kunihiko Hiraishi received from the Tokyo Institute of Technology the B. E. degree in 1983, the M. E. degree in 1985, and D. E. degree in 1990. He is currently a professor at School of Information Science, Japan Advanced Institute of Science and Technology. His research interests include discrete event systems and formal verification. He is a member of the IEEE, IPSJ, and SICE.