Computing attractors of large-scale asynchronous boolean networks using minimal trap spaces

Van-Giang Trinh, Kunihiko Hiraishi, Belaid Benhamou

Computing Attractors of Large-Scale Asynchronous Boolean Networks Using Minimal Trap Spaces

Van-Giang Trinh*
LIS, AIX-Marseille Université
Marseille, France
giang.trinh91@gmail.com

Kunihiko Hiraishi
School of Information Science, Japan Advanced Institute of Science and Technology
Nomi, ishikawa, Japan
hira@jaist.ac.jp

Belaid Benhamou
LIS, AIX-Marseille Université
Marseille, France
belaid.benhamou@univ-amu.fr

ABSTRACT

Boolean Networks (BNs) play a crucial role in modeling, analyzing, and controlling biological systems. One of the most important problems on BNs is to compute all the possible attractors of a BN. There are two popular types of BNs, Synchronous BNs (SBNs) and Asynchronous BNs (ABNs). Although ABNs are considered more suitable than SBNs in modeling real-world biological systems, their attractor computation is more challenging than that of SBNs. Several methods have been proposed for computing attractors of ABNs. However, none of them can robustly handle large and complex models. In this paper, we propose a novel method called mtsNFVS for exactly computing all the attractors of an ABN based on its minimal trap spaces, where a trap space is a subspace of state space that no path can leave. The main advantage of mtsNFVS lies in opening the chance to reach easy cases for the attractor computation. We then evaluate mtsNFVS on a set of large and complex real-world models with crucial biologically motivations as well as a set of randomly generated models. The experimental results show that mtsNFVS can easily handle large-scale models and it completely outperforms the state-of-the-art method CABEAN as well as other recently notable methods.

KEYWORDS

biological system, asynchronous Boolean network, attractor, minimal trap space, negative feedback vertex set

ACM Reference Format:
Van-Giang Trinh, Kunihiko Hiraishi, and Belaid Benhamou. 2022. Computing Attractors of Large-Scale Asynchronous Boolean Networks Using Minimal Trap Spaces. In *13th ACM International Conference on Bioinformatics, Computational Biology and Health Informatics, August 07–10, 2022, Chicago, IL, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Boolean Networks (BNs) are simple but efficient mathematical formalism for modeling, analyzing, and controlling complex biological systems (e.g., gene regulatory networks, signal transduction networks) [22, 39, 40]. Beyond systems biology, BNs have widely been applied to various other areas, such as, mathematics, neural networks, social modeling, robotics, and computer science (see, e.g., [14, 39, 49]). Besides a plenty of applications, BNs are also an interesting mathematical object that has recently attracted various work in both theoretical and computational aspects [39].

Attractor detection in BNs is difficult and interesting in theory but also has a plenty of applications in many areas [1]. 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 is referred to as *attractors*. An attractor of a BN is a set of states such that the BN cannot escape from this set once entered it. In the biological context, attractors of a BN are linked to phenotypes [22] or functional cellular states (e.g., proliferation, apoptosis, or differentiation) [19]. Hence, analysis of attractors could provide new insights into systems biology [3] (e.g., the origins of cancers [4], SARS-CoV-2 [31], HIV [32]), which play an important role in the development of new drugs [20]. Furthermore, attractor detection also gives a starting point for many control approaches for biological systems [7, 43]. To sum up, attractor detection is of great importance in analyzing and controlling biological systems modeled as BNs.

There are two main types of BNs usually used for modeling biological systems: Synchronous BNs (SBNs) [22] and Asynchronous BNs (ABNs) [45]. The updating scheme of SBNs is that all the nodes are updated simultaneously at each time step [12]. The updating scheme of ABNs is that only one node is randomly and uniformly selected in order to be updated at each time step [12]. In biology, the updating process of each gene may spend various time from fractions of a second to hours [38]. Moreover, the information on time scales of components is usually lacking [38]. Hence, ABNs are considered more suitable [38, 45] for representing various time scales as well as dealing with the lack of knowledge on time scales. However, the dynamics of an ABN is generally more complex than that of its SBN counterpart, making the analysis of this ABN more challenging [12, 38].

Several methods have been proposed for computing attractors of ABNs, including symbolic-based methods [5, 6, 12, 13, 53], structure-based methods [24, 36, 42, 51], decomposition-based methods [26, 44]. It is noted that there are also several methods [15, 23, 24, 36, 51] for approximating attractors of ABNs. Obviously, they however cannot guarantee finding exactly all the attractors of an ABN. To our best knowledge, all the above-mentioned methods do not satisfactorily handle large and complex models, i.e., the ones that have

BCB '22, August 07–10, 2022, Chicago, IL, USA

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

many nodes (e.g., hundreds of nodes and beyond) and many interactions among the nodes. The best state-of-the-art methods [26, 51] are not yet enable to robustly work with such ABN models.

Recently, an efficient method called iFVS-ABN [17] has been proposed for exactly computing all the attractors of an ABN. iFVS-ABN first computes a Negative Feedback Vertex Set (NFVS) of the interaction graph of the ABN that is a signed directed graph that expresses the effects (positive or negative) among the nodes. Based on the chosen NFVS, iFVS-ABN systematically removes arcs in the State Transition Graph (STG) of the ABN to get a candidate set of states. Then, iFVS-ABN uses the reachability analysis on the ABN to filter out this candidate set. Note that iFVS-ABN also uses a preprocessing step called Preprocessing SSF to reduce the number of times the reachability in ABNs is checked. The approach of iFVS-ABN seems to be very promising and its prototype implementation significantly outperforms the previous methods including genYsis [12], CABEAN [26], and FVS-ABN [46] (the predecessor of iFVS-ABN). However, the crucial issue of iFVS-ABN is that it still must perform the reachability analysis in most cases [17]. Since the reachability in ABNs is PSPACE-complete [9] and there is no reachability analysis method that is robustly efficient for large models [17], the issue may drastically reduce the efficiency of iFVS-ABN.

In this paper, we propose a novel method named mtsNFVS for exactly computing all the attractors of an ABN. The method exploits the advantages of the efficient method [23] for computing minimal trap spaces of the ABN and the NFVS-based approach of iFVS-ABN [17]. In principle, similar to iFVS-ABN, mtsNFVS also relies on NFVSs and the reducing dynamics to get a candidate set of states, then filters out this set to get a resulting set that exactly covers the set of attractors of the ABN. However, by using minimal trap spaces mtsNFVS can open a chance to reach easy cases for the reachability analysis in the filtering process, which are generally unable in iFVS-ABN. In addition, we then propose several algorithmic improvements to several common constituent tasks between mtsNFVS and iFVS-ABN, such as, the computation of the NFVS, the computation of the candidate set, and Preprocessing SSF. These algorithmic improvements are key factors for making the chance opened by mtsNFVS effective.

We then use a set of large and complex real-world models obtained from the literature in the field of systems biology and compare the prototype implementation of mtsNFVS to the state-of-the-art method CABEAN [26, 43] as well as other recently notable methods including AEON [6], PyBoolNet [23, 24], pystablemotifs [35, 36], and iFVS-ABN [17]. The experimental results show that mtsNFVS completely outperforms all the other methods. In particular, mtsNFVS is the only method that can consistently handle the two most complex models of the benchmark set. Furthermore, we also conduct an experiment on randomly generated models and obtain the similar experimental conclusion.

The rest of this paper is organized as follows: Section 2 introduces the basic concepts including Boolean networks, attractors, interaction graphs, negative feedback vertex sets, and minimal trap spaces. Section 3 presents the details of the proposed method (i.e., mtsNFVS). Section 4 reports the experimental results for evaluating the efficiency of mtsNFVS. Finally, Section 5 concludes the paper and draws future work.

2 PRELIMINARIES

Let $\mathbb{N}$ denote the set of natural numbers. Denote by $\mathbb{N}^+$ the set $\mathbb{N}\backslash\{0\}$ and by $\mathbb{N}^+_{\leq k}$ the set $\{i \in \mathbb{N}^+ : i \leq k\}$. $\mathbb{B} := \{T \equiv 1, F \equiv 0\}$ denotes the Boolean domain. $|A|$ denotes the cardinality of a set A.

2.1 Boolean networks and their attractors

A Boolean Network (BN) [14] is defined as a pair (V, F), where $V = \{x_1, ..., x_n\}$ ($n \in \mathbb{N}^+$) is the set of nodes and $F = \{f_1, ..., f_n\}$ is the set of Boolean functions. The size of a BN is characterized by its number of nodes n. Each node x_i is identified as a Boolean variable, and is associated with a Boolean function $f_i : \mathbb{B}^{|IN(f_i)|} \to \mathbb{B}$, where $IN(f_i)$ is the set of input nodes of f_i. The number of edges of a BN is defined as $\sum_{i=1}^{n} |IN(f_i)|$. Then, the average connectivity of a BN is defined as the number of edges per node, i.e., $\frac{\sum_{i=1}^{n}|IN(f_i)|}{n}$.

Let $x_i(t) \in \mathbb{B}$ and $x(t) = (x_1(t), ..., x_n(t)) \in \mathbb{B}^n$ denote the state of node x_i and the state of the BN at time t, respectively. At each time step, node x_i can update its state by

$$x_i(t+1) = f_i(x(t)).$$

For simplicity, we use the notation $f_i(x(t))$ even if $IN(f_i) \subset V$. An updating scheme of a BN specifies the way that the nodes of the BN update their states through time evolution [14]. Following the updating scheme, the BN transits from a state to another state (possibly identical). This transition is called the *state transition*.

An Asynchronous Boolean Network (ABN) [18] can be seen as the most popular BN model. The updating scheme of an ABN is fully asynchronous. That is, at each time step, only one node is nondeterministically selected in order to be updated. The whole dynamics of an ABN can be captured by a State Transition Graph (STG) that is a directed graph in which each node corresponds to a state of the ABN and each arc corresponds to a state transition between two states (possibly identical). The STG of an ABN of size n has 2^n nodes and up to $n \times 2^n$ arcs, making the analysis of the ABN more difficult [38].

Attractors are key dynamical behavior of a BN [38]. An attractor of a BN is defined as a set of states satisfying any state in this set can reach any state in this set and cannot reach any other state that is not in this set [26]. Then, we can classify two main types of attractors: singleton and cyclic attractors. A singleton attractor (or a fixed point) consists of only one state. A cyclic attractor consists of at least two states, and is formed by overlapping one or more cycles of states. In general, an attractor of an ABN is equivalent to a bottom Strongly Connected Component (SCC) of the STG of this ABN [12]. Therefore, naive approaches for computing attractors of an ABN (e.g., explicitly building the STG and then applying graph algorithms) are intractable for large networks (e.g., $n \geq 20$) [8].

Example 1. *We give a BN $\mathcal{N} = \{V, F\}$, where $V = \{x_1, x_2\}$ and $F = \{f_1, f_2\}$ with $f_1 = (x_1 \wedge x_2) \vee (\neg x_1 \wedge \neg x_2)$, $f_2 = (x_1 \wedge x_2) \vee (\neg x_1 \wedge \neg x_2)$. Herein, $\wedge$, $\vee$, and $\neg$ denote the CONJUNCTION, DISJUNCTION, and NEGATION logical operators, respectively.*

Let us consider the BN shown in Example 1. Figure 1a shows the STG of its ABN counterpart. As we can see, the ABN has one fixed point ($\{11\}$) and one cyclic attractor ($\{00, 01, 10\}$).

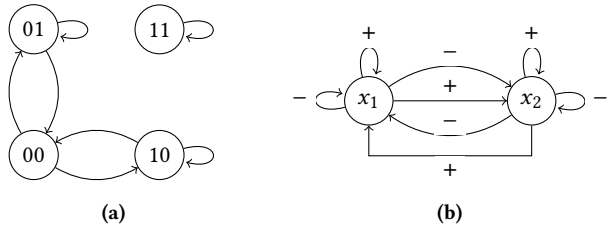

Figure 1: (a) STG of the ABN counterpart of the BN shown in Example 1. (b) Interaction graph of the BN shown in Example 1.

2.2 Interaction graphs

The *interaction graph* of a BN depicts the qualitative interactions between nodes and is usually represented as a signed directed graph on the set of nodes. An arc (x_j, x_i) indicates that the evolution of node x_i depends on the evolution of node x_j. In addition, an arc can be either positive or negative. The formal and detailed definition for the interaction graph of a BN can be found at [34]. Furthermore, the computation of the interaction graph of a BN is often fast [17, 34].

Let IG be a signed directed graph. A positive (resp. negative) cycle of IG is an elementary directed cycle that contains an even (resp. odd) number of negative arcs. The length of a cycle is the number of arcs it involves[1]. Then, a *Feedback Vertex Set* (FVS) of IG is a set of vertices U that intersects every cycle of IG. In other words, IG becomes acyclic after removing the vertices in U from IG. A *Negative Feedback Vertex Set* (NFVS) of IG is a set of vertices U^- that intersects every negative cycle of IG. In other words, IG has no negative cycle after removing the vertices in U^- from it. Clearly, an FVS is also an NFVS. The problem of finding a minimum NFVS has been proved NP-complete [28]. The problem of finding a minimum FVS has also been proved NP-complete [21].

As an example, we consider the BN $\mathcal{N}$ shown in Example 1. Figure 1b shows the interaction graph of $\mathcal{N}$. Arrows labeled with symbol "+" denote positive arcs, whereas arrows labeled with symbol "−" denote negative arcs. This interaction graph has two negative cycles of length 1 ($x_1 \xrightarrow{-} x_1$ and $x_2 \xrightarrow{-} x_2$) and two negative cycles of length 2 ($x_1 \xrightarrow{-} x_2 \xrightarrow{+} x_1$ and $x_2 \xrightarrow{-} x_1 \xrightarrow{+} x_2$). Then, it has one NFVS $\{x_1, x_2\}$, which is also the minimum one.

2.3 Minimal trap spaces

First, we prepare several notions as follows. Let $S_{\mathcal{A}} = \mathbb{B}^n$ denote the state space of an ABN $\mathcal{A}$ that consists of all possible 2^n states. The forward image of a state x with respect to node $x_i \in V$ (denoted by $FI_i^{\mathcal{A}}(x)$) is defined to be the state y such that $x_j = y_j$ for all $j \in \mathbb{N}_{\leq n}^+ \backslash \{i\}$ and $y_i = f_i(x)$. Generally, the forward image of a state set A with respect to node $x_i \in V$ is defined as $FI_i^{\mathcal{A}}(A) := \bigcup_{x \in A} FI_i^{\mathcal{A}}(x)$. Following the updating scheme of ABNs, (x, y) is an arc of the STG of $\mathcal{A}$ if and only if there exists $i \in \mathbb{N}_{\leq n}^+$ such that $y = FI_i^{\mathcal{A}}(x)$. We also use $x \xrightarrow{\mathcal{A}} y$ (or simply $x \to y$ whenever the context is clear) to denote this arc.

[1] A self arc is considered as a cycle of length 1.

A non-empty set $T \subseteq S_{\mathcal{A}}$ is a *trap set* of the STG of $\mathcal{A}$ if and only if for every $x \in T$ and $y \in S_{\mathcal{A}}$ with $x \to y$ it holds that $y \in T$. By this definition, an attractor of $\mathcal{A}$ is equivalent to an inclusion-wise minimal trap set of its STG [23]. Consequently, every trap set contains at least one minimal trap set and therefore at least one attractor. A subspace m of $S_{\mathcal{A}}$ is characterized by its fixed variables (denoted by D_m) and free variables. This subspace can be specified by an assignment $m : D_m \to \mathbb{B}$ where $D_m \subseteq V$ and $m(u)$ is the value of node $u \in D_m$. The remaining variables of $\mathcal{A}$, $V \backslash D_m$, are said to be free, i.e., they can receive any Boolean value. We specify subspaces like states but use in addition the symbol $\star$ to indicate that a variable is free. In addition, a subspace m also refers to the set of states $S[m] := \{s \in S_{\mathcal{A}} | \forall x_i \in D_m : s_i = m(x_i)\}$. For example, $m = \star\star 1$ means that $D_m = \{x_3\}$, $m(x_3) = 1$, and refers to the set of states $\{001, 011, 101, 111\}$. Let $S_{\mathcal{A}}^{\star}$ denote the set of all possible subspaces. Note that $\left|S_{\mathcal{A}}^{\star}\right| = 3^n$ and $S_{\mathcal{A}} \subset S_{\mathcal{A}}^{\star}$ [23].

A *trap space* is defined as a subspace that is also a trap set. Then, we define a partial order $<$ on $S_{\mathcal{A}}^{\star}$ as: $m < m'$ if and only if $S[m] \subseteq S[m']$ and $S[m] \neq S[m']$. Consequently, a trap space m is minimal if and only if there is no trap space $m' \in S_{\mathcal{A}}^{\star}$ such that $m' < m$. For example, the ABN counterpart of the BN shown in Example 1 has all two trap spaces, $m_1 = 11$ and $m_2 = \star\star$. Since $m_1 < m_2$, m_1 is a minimal trap space of the ABN.

3 NEW METHOD

We here propose a new method named mtsNFVS for exactly computing all the attractors of an ABN. This method exploits the advantages of the minimal trap space computation method [23] and the approach of the NFVS-based method iFVS-ABN [17]. Hereafter, we shall present the main idea as well as several key constituent tasks in mtsNFVS.

3.1 Main idea

The main idea of mtsNFVS is similar to that of iFVS-ABN but using minimal trap spaces of the ABN to guide the attractor computation to some easy cases. Specifically, the general description of mtsNFVS is given in Algorithm 1. For convenience, we first introduce some new notations as follows:

- $IG(\mathcal{N})$ denotes the interaction graph of a BN $\mathcal{N}$;
- $G(\mathcal{N})$ denotes the STG of an BN $\mathcal{N}$;
- $F(G)$ denotes the set of fixed points of an STG G;
- $R_{U,B}$ denotes the operation of systematically removing arcs from an STG with respect to a set U of nodes and a set B of retained Boolean values corresponding to the nodes in U (see [17] for more details);
- $R_{U,B}(G)$ denotes the reduced STG obtained by applying $R_{U,B}$ to an STG G;
- $S[M] = \bigcup_{m \in M} S[m]$ is the set of states represented by a set M of minimal trap spaces;
- We say that a set F of states covers a set A of attractors if and only if F intersects every attractor of A (formally, $F \cap att \neq \emptyset, \forall att \in A$);
- We say that F *one-to-one* covers the set A of attractors if and only if F covers A and $|F| = |A|$.

Algorithm 1 mtsNFVS

Require: An ABN $\mathcal{A}$.
Ensure: A set A of states of $\mathcal{A}$.
► Lines 1-7: Compute the set of minimal trap spaces and the candidate set of states
1: Find an NFVS $U^- = \{x_{i_1}, ..., x_{i_k}\}$ of $IG(\mathcal{A})$
2: Choose a set $B^- = \{b_{i_1}, ..., b_{i_k}\}$ of retained Boolean values corresponding to the nodes in U^-
3: $M \leftarrow$ the set of minimal trap spaces of $\mathcal{A}$
4: $F \leftarrow F(R_{U^-,B^-}(G(\mathcal{A})))$
5: $A \leftarrow \emptyset$
6: $A_{in} \leftarrow \emptyset$
7: $A_{out} \leftarrow \emptyset$
► Lines 8-20: Compute attractors inside each minimal trap space
8: **for all** $m \in M$ **do**
9: $F_m \leftarrow S[m] \cap F$
10: $F \leftarrow F \backslash F_m$
11: $A_m \leftarrow \emptyset$
12: Perform Preprocessing SSF to shrink the set F_m if needed
13: **while** $F_m \neq \emptyset$ **do**
14: Remove a state s from F_m
15: **if** ABNReach$(\mathcal{A}, s, A_m \cup F_m) = false$ **then**
16: $A_m \leftarrow A_m \cup \{s\}$
17: **end if**
18: **end while**
19: $A_{in} \leftarrow A_{in} \cup A_m$
20: **end for**
► Lines 21-29: Compute attractors outside the minimal trap spaces
21: $F \leftarrow F \backslash S[M]$
22: Perform Preprocessing SSF to shrink the set F if needed
23: $F \leftarrow F \backslash S[M]$
24: **while** $F \neq \emptyset$ **do**
25: Remove a state s from F
26: **if** ABNReach$(\mathcal{A}, s, S[M] \cup A_{in} \cup A_{out} \cup F) = false$ **then**
27: $A_{out} \leftarrow A_{out} \cup \{s\}$
28: **end if**
29: **end while**
► Lines 30-31: Return the set of states corresponding to all the inside and the outside attractors
30: $A \leftarrow A_{in} \cup A_{out}$
31: **return** A

In general, mtsNFVS uses an NFVS of the ABN $\mathcal{A}$ to get a candidate set of states (Line 4 of Algorithm 1). This step is similar to that in iFVS-ABN. However, mtsNFVS calculates the set of minimal trap spaces of $\mathcal{A}$ instead of the set of fixed points of $\mathcal{A}$ (Line 3 of Algorithm 1). Each minimal trap space contains at least one attractor of $\mathcal{A}$ and minimal trap spaces are mutually disjoint. Therefore, for each minimal trap space m, mtsNFVS gets a candidate set F_m of states, and then obviously excludes F_m from F (Lines 9-10 of Algorithm 1). Since F covers all the attractors of $\mathcal{A}$ (see Theorem 2 of [17]), F_m must cover all the attractors contained in minimal trap space m. If F_m contains many states, mtsNFVS can use Preprocessing SSF to shrink it (Line 12 of Algorithm 1). Now, mtsNFVS performs the filtering process (as that in [17, 46]) on the candidate set F_m to get all the attractors contained in m (Lines 13-18 of Algorithm 1). Herein, ABNReach is the efficiently *exact* algorithm proposed in [17] for checking the reachability in ABNs. The parameters of this algorithm include the considered ABN, the initial state, and the set of target states (e.g., $\mathcal{A}$, s, and $A_m \cup F_m$ in Line 15 of Algorithm 1, respectively). Note that in each minimal trap space m, Preprocessing SSF and the filtering process can be performed on the reduced ABN $\mathcal{A}'$ instead of the original ABN $\mathcal{A}$. $\mathcal{A}'$ is obtained by fixing the fixed nodes of m in $\mathcal{A}$, and then propagating the fixed values to the Boolean functions of the remaining nodes (see [23] for the reduction technique using minimal trap spaces). Since $\mathcal{A}'$ is significantly smaller than $\mathcal{A}$ in most cases [23, 36], the use of the reduction technique may (potentially) reduce significantly the computational burden for each minimal trap space.

As mentioned in [23], there may be some attractors that are outside of any minimal trap space of $\mathcal{A}$. Hence, mtsNFVS needs to process the remaining part of F. If this part contains many states, mtsNFVS can use Preprocessing SSF to shrink it (Line 22 of Algorithm 1). Note that after Preprocessing SSF, F may contain some states in $S[M]$. Therefore, mtsNFVS needs to exclude these states from F (Line 23 of Algorithm 1). mtsNFVS then performs the filtering process on the current candidate set F (Lines 24-29 of Algorithm 1). There is a difference to the filtering process for each minimal trap space. When checking the reachability in $\mathcal{A}$, the target set can be expanded to $S[M] \cup A_{in} \cup A_{out} \cup F$ instead of only $A_{in} \cup A_{out} \cup F$ as in [17, 46] (Line 26 of Algorithm 1). The reason is that if a state s reaches a state $s' \in S[M]$ in the STG of $\mathcal{A}$, then s must reach at least one attractor contained in M, consequently s must reach a state in A_{in}. This expansion may reduce the time for checking the reachability in $\mathcal{A}$ (especially in the case of reachable) because $S[M] \cup A_{in} \cup A_{out} \cup F$ may be potentially much larger than $A_{in} \cup A_{out} \cup F$. Finally, mtsNFVS returns the set A of states that one-to-one covers the set of attractors of $\mathcal{A}$.

For illustration, we here show a running example for mtsNFVS. Let us consider the ABN counterpart $\mathcal{A}$ of the BN shown in Example 1. As shown in Figure 1a, $\mathcal{A}$ has one fixed point ($\{11\}$) and one cyclic attractor ($\{00, 01, 10\}$). The interaction graph of $\mathcal{A}$ has one NFVS (also the minimum one) $\{x_1, x_2\}$. Next, assume that mtsNFVS chooses $U^- = \{x_1, x_2\}$ and $B^- = \{b_1, b_2\} = \{0, 0\}$. We then have $F(R_{U^-,B^-}(G(\mathcal{A}))) = \{00, 11\}$, $M = \{m_1\}$ where $m_1 = 11$, $S[M] = S[m_1] = \{11\}$. For m_1, we get $F_m = \{11\}$. Since F_m has only one state, we do not need to proceed Preprocessing SSF and simply add this state to the set A_m. After finishing Line 19 of Algorithm 1, we have $A_{in} = \{11\}$ and $F = \{00\}$. As we can see, the cyclic attractor $\{00, 01, 10\}$ is outside of any minimal trap space of $\mathcal{A}$ but it is still covered by F. The latter part of mtsNFVS (Lines 21-29 of Algorithm 1) guarantees to compute all the attractors of $\mathcal{A}$. Now, mtsNFVS performs the filtering process on F. Since 00 (i.e., s) does not reach $\{11\}$ (i.e., $S[M] \cup A_{in} \cup A_{out} \cup F$) in $G(\mathcal{A})$, mtsNFVS adds 00 to the set A_{out}. Finally, mtsNFVS returns $A = \{11, 00\}$ where 11 corresponds to the fixed point $\{11\}$ and 00 corresponds to the cyclic attractor $\{00, 01, 10\}$.

Next, we show the correctness of mtsNFVS as in Theorem 3.1.

Theorem 3.1. *Algorithm 1 exactly finds all the attractors of an ABN.*

Proof. After finishing Line 4 of Algorithm 1, F covers all the attractors of $\mathcal{A}$ (*) following Theorem 2 of [17] that shows that every attractor of $\mathcal{A}$ always contains at least one fixed point of the reduced STG with respect to U^- and B^-. Hence, F_m (also $A_m \cup F_m$) must cover all the attractors of $\mathcal{A}$ contained in minimal trap space m. Since Preprocessing SSF and the filtering process always preserve this property (see the proof of Theorem 3 of [17] that shows the correctness of iFVS-ABN), A_m finally one-to-one covers the set of all attractors of $\mathcal{A}$ contained in m. As a consequence, after finishing Line 20 of Algorithm 1, A_{in} one-to-one covers all the attractors inside the minimal trap spaces of $\mathcal{A}$ (**).

A state in $S[m]$ always reach in $G(\mathcal{A})$ only the attractors inside minimal trap space m of $\mathcal{A}$. As a consequence, every state in $S[M]$ always reaches in $G(\mathcal{A})$ the states in A_{in} (***). After finishing Line 21 of Algorithm 1, F (also $A_{out} \cup F$) covers only all the attractors outside any minimal trap space of $\mathcal{A}$ because (*) and (***) holds. Clearly, after finishing Preprocessing SSF and Line 23 of Algorithm 1, this property is preserved. In addition, state s reaches $A_{in} \cup A_{out} \cup F$ in $G(\mathcal{A})$ if and only if s reaches $S[M] \cup A_{in} \cup A_{out} \cup F$ in $G(\mathcal{A})$ because (***) holds. Therefore, the filtering process also preserves the property. Then, after finishing Line 29 of Algorithm 1, A_{out} finally one-to-one covers all the attractors outside any minimal trap space of $\mathcal{A}$ (****).

From (**) and (****), we can conclude that the resulting set A one-to-one covers the set of attractors of $\mathcal{A}$. □

Finally, we discuss the advantages of mtsNFVS as compared to PyBoolNet (i.e., the method for approximating the attractors of an ABN [23, 24]) and iFVS-ABN. First, PyBoolNet relies on only the set of minimal trap spaces of an ABN to get the set of approximations. For each minimal trap space, PyBoolNet uses random walks to get an approximation (a state) that is expected belonging to an attractor contained in the minimal trap space. However, there may be some attractors that are outside of any minimal trap space or one minimal trap space may contain more than one attractor. Hence, we have no chance to get such attractors with PyBoolNet. On the other hand, mtsNFVS always guarantees to compute all the attractors of the ABN (see Theorem 3.1). Second, iFVS-ABN always must perform the reachability analysis unless 1) the ABN has only fixed points and Preprocessing SSF makes the number of candidates equal 0 [17] or 2) the ABN has only one cyclic attractor and Preprocessing SSF makes the number of candidates equal 1 [17]. Hence, iFVS-ABN has no chance to avoid the reachability analysis in most cases. On the other hand, mtsNFVS has a chance to avoid the reachability analysis if the number of attractors of the ABN is equal to the number of minimal trap spaces of the ABN, which occurs in most cases [23, 35, 36]. In the case, if Preprocessing SSF in each minimal trap space (Line 12 of Algorithm 1) makes the number of candidates equal 1 and Preprocessing SSF in the remaining candidate part (Line 22 of Algorithm 1) makes the number of candidates equal 0, then mtsNFVS does not need to check the reachability in ABNs. Furthermore, the expansion of the target set in mtsNFVS (Line 26 of Algorithm 1) may reduce the time for the reachability analysis in the case of reachable. However, to make the above advantages as compared to iFVS-ABN effective, we also need to propose algorithmic improvements to several common constituent tasks of mtsNFVS and iFVS-ABN such as Preprocessing SSF. We present the proposed algorithmic improvements in the following subsections.

3.2 Computing negative feedback vertex sets and minimal trap spaces

The first step of mtsNFVS is to find an NFVS U^- of the interaction graph of the ABN (see Line 1 of Algorithm 1). We note that using a smaller NFVS would open a chance to get a smaller candidate set [17] and the problem of finding a minimum NFVS of a signed directed graph is NP-complete [21]. In [17], iFVS-ABN uses a simple greedy algorithm called findNFVS for finding an (not necessarily minimum) NFVS. The seed set used in this greedy algorithm is the FVS of the ABN obtained by applying the simple greedy algorithm by [46], which does not guarantee to return a minimum FVS, to the interaction graph. Clearly, using a smaller seed can open a chance to get a smaller NFVS. Hence, we here apply the algorithm by [11] for approximately computing an FVS of the interaction graph. We use an implementation of this algorithm that is publicly available at https://github.com/jgtz/FVS_python3. The usefulness of the updated algorithm for finding an (not necessarily minimum) NFVS shall be shown in our benchmarks presented in Section 4.

Regarding the computation of minimal trap spaces (see Line 3 of Algorithm 1), we simply use the Answer Set Programming (ASP) based method by [23], which is integrated into PyBoolNet [24]. Note that, there may be other methods for computing minimal trap spaces of an ABN such as the Integer Linear Programming (ILP) based method by [23], which is also integrated into [24]. However, in [23] the ASP-based method has been shown more time-efficient than the ILP-based method.

3.3 Computing fixed points of the reduced state transition graph

Similar to iFVS-ABN [17], mtsNFVS also needs to compute the set of fixed points of the reduced STG with respect to a set of nodes U^- and a set of Boolean values B^- (i.e., $F(R_{U^-,B^-}(G(\mathcal{A})))$ as shown in Line 4 of Algorithm 1). In [17], $F(R_{U^-,B^-}(G(\mathcal{A})))$ is computed by using Binary Decision Diagrams (BDDs) [48] or Satisfiability (SAT) solvers [10]. Both the techniques are inefficient for large networks especially the networks comprising complex Boolean functions [17, 46]. Hence, we need a more efficient method.

We first construct an ABN (denoted by $\mathcal{A}^{red}$) such that its set of fixed points is identical to $F(R_{U^-,B^-}(G(\mathcal{A})))$ (see Theorem 3.2). $\mathcal{A}^{red}$ includes the set of nodes of $\mathcal{A}$ and its set of Boolean functions is given by:

$$\begin{cases} f_i^{red} = f_i & \text{if } x_i \notin U^-; \\ f_i^{red} = [(x_i \leftrightarrow b_i) \wedge b_i] \vee [\neg(x_i \leftrightarrow b_i) \wedge f_i] & \text{if } x_i \in U^-; \end{cases}$$

where $\leftrightarrow$ denotes the BI-IMPLICATION logical operator.

Theorem 3.2. *The set of fixed points of $\mathcal{A}^{red}$ is identical to the set of fixed points of the reduced STG of $\mathcal{A}$ with respect to U^- and B^-.*

Proof. It is important to note that a state x in $G(\mathcal{A})$ will become a fixed point of $R_{U^-,B^-}(G(\mathcal{A}))$ if and only if (1) $f_i(x) = x_i, \forall x_i \notin U^-$ and (2) $x_i = b_i$ or $x_i = 1 - b_i \wedge f_i(x) = x_i, \forall x_i \in U^-$ [17].

Suppose that x is a fixed point of $\mathcal{A}^{red}$. Then, $f_i(x) = f_i^{red}(x) = x_i$ for $\forall x_i \notin U^-$. We consider the case that $x_i \in U^-$. If $x_i = b_i$,

then $f_i^{red} = b_i = x_i$; leading there is no constraint on $f_i(x)$. If $x_i = 1 - b_i$, then $f_i(x) = f_i^{red}(x) = x_i$. Hence, x is also a fixed point of $R_{U^-,B^-}(G(\mathcal{A}))$.

Suppose that x is a fixed point of $R_{U^-,B^-}(G(\mathcal{A}))$. Then, $f_i^{red}(x) = f_i(x) = x_i$ for $\forall x_i \notin U^-$. We consider the case that $x_i \in U^-$. If $x_i = 1 - b_i$, then $f_i(x) = x_i$; leading to $f_i^{red}(x) = f_i(x) = x_i$. If $x_i = b_i$, then there is no constraint on $f_i(x)$ but $f_i^{red}(x) = b_i = x_i$. Hence, x is also a fixed point of $\mathcal{A}^{red}$. □

From Theorem 3.2, we now can compute $F(R_{U^-,B^-}(G(\mathcal{A})))$ by computing the set of fixed points of $\mathcal{A}^{red}$. The ASP-based method by [23] has been recognized very efficient for computing the set of fixed points of an ABN [23, 24, 35, 36]. Note that the ILP-based method by [2] and the algebraic-based method by [50] are also very efficient methods for this task; but they are designed specifically for special classes of BNs (N-K models with $K = 2$ and AND-NOT models, respectively). Hence, we here use the ASP-based method for computing the set of fixed points of $\mathcal{A}^{red}$. Its usefulness shall be shown in our benchmarks presented in Section 4.

3.4 Improving Preprocessing SSF

For convenience, we briefly recall the description of Preprocessing SSF that was first introduced in [46]. Preprocessing SSF aims at shrinking the candidate set F. At each iteration, Preprocessing SSF randomly chooses a node x_i, then updates F by its forward image with respect to node x_i (i.e., $F \leftarrow FI_i^{\mathcal{A}}(F)$). The number of iterations of Preprocessing SSF is specified by the parameter I_MAX, which can be empirically set. It is important to note that in [17, 46] $FI_i^{\mathcal{A}}(F)$ is computed based on the restricted transition system of $\mathcal{A}$ with respect to the set F encoded as a BDD of $2n$ variables [12, 16]. Hence, the computation of $FI_i^{\mathcal{A}}(F)$ may be too long even intractable for large and complex networks.

Since the reachability in ABNs is PSPACE-complete in theory and may take extremely long time in practice, Preprocessing SSF usually sets I_MAX large enough with the expectation that the number of candidates for the filtering process is as small as possible. Accordingly, the time for Preprocessing SSF may become too long. To improve the efficiency of Preprocessing SSF (consequently, the efficiency of mtsNFVS), we propose two algorithmic enhancements for Preprocessing SSF in mtsNFVS as follows.

First, we propose a new way for computing $FI_i^{\mathcal{A}}(F)$. Instead of using the restricted transition system, mtsNFVS computes the forward image of each state x in F. $FI_i^{\mathcal{A}}(x)$ is easily computed by only changing the value of node x_i to $f_i(x)$. Clearly, the number of forward images needed to be computed at each iteration of Preprocessing SSF is $|F|$. Hence, the new way is simple but maybe very efficient because $|F|$ drastically decreases in most cases [17, 46]. Note that the new way can apply to both Preprocessing SSF in each minimal trap space and Preprocessing SSF in the remaining candidate part (see Subsection 3.1).

Second, for Preprocessing SSF in the remaining candidate part, we try to early exclude from F some states that cannot belong to an attractor outside the minimal trap spaces of the ABN. The set of states referred by the minimal trap spaces (i.e., $S[M]$) usually contains much more states than F. Therefore, it is likely possible that several states of F will be covered by $S[M]$ after only a small number of iterations. Since a state in $S[M]$ cannot reach an outside attractor, we can early exclude the states in $S[M]$ from F. Specifically, we divide Preprocessing SSF into many spans. Each span includes a given number of iterations (i.e., the length of a span). When finishing each span, we simply perform $F \leftarrow F \backslash S[M]$. For simplicity, we set the same length (empirically n) for all spans.

Both the proposed enhancements may reduce the computational time of Preprocessing SSF. In particular, they allow us to set I_MAX much larger (empirically $20000 \times n$), thus likely to get a smaller candidate set even to early encounter the best cases (i.e., $|F| = 1$ for Preprocessing SSF in each minimal trap space and $|F| = 0$ for Preprocessing SSF in the remaining candidate part). The usefulness of the two enhancements shall be justified by our benchmarks presented in Section 4.

4 EVALUATION

To evaluate the effectiveness of the proposed method mtsNFVS, we conducted experiments on both real-world biological models and randomly generated models. We compared mtsNFVS with five previously notable methods including CABEAN [26, 43], AEON [6], PyBoolNet [23, 24], pystablemotifs [35, 36], and iFVS-ABN [17]. To the best of our knowledge, all these previous methods are the most recent and advanced tools targeting the detection of non-trivial attractors in ABNs. Moreover, there has been no complete comparison among them.

4.1 Experimental results on real-world models

We selected seven real-world models, which are large and complex (i.e., high average connectivity), from the literature. We here give a brief description of the models. The structure information of the models can be found in Table 1.

- The T-LGL network models the T cell large granular lymphocyte (T-LGL) survival signaling network that features a clonal expansion of antigen-primed, competent, and cytotoxic T lymphocytes [52].
- The CACC model describes the development of colitis-associated colon cancer by integrating the extracellular microenvironment and intracellular signalling pathways, helping to obtain a more systematic understanding of inflammation-associated tumourigenesis as well as to identify novel therapeutic approaches [25].
- The AD model is a relevant mathematical model of the neuronal molecular regulatory network for Alzheimer's disease, with the goal of enabling researchers to gain a better mechanistic understanding of Alzheimer's disease pathological dynamics at a molecular-regulation level and systematically investigate candidate molecular targets for their ability to alter the levels of pathogenic proteins [33].
- The IL-6 model is a comprehensive large-scale network model whose analysis helps to uncover general topological features and to make testable predictions on the stimulus-response behaviour of the IL-6 signalling network, which is crucially involved in the regulation of a multitude of physiological processes, in particular coordinating the immune response upon bacterial infection and tissue injury [37].

- The CELL CYCLE 2019 network, a Boolean model of model growth factor signaling, can reproduce PI3K oscillations and link them to cell cycle progression and apoptosis; thus is an important starting point for the predictive modeling of cell fate decisions that include AKT1-driven senescence, as well as the non-intuitive effects of drugs that interfere with mitosis [41].
- The SIPC model considers the major signalling pathways known to be deregulated, helping to better understand the mechanisms of tumorigenesis and possible treatment responses for prostate cancer, the second most occurring cancer in men worldwide [27].
- The CASCADE 3.0 model is the most recent Boolean model of cancer cell lines, which demonstrates the potential of logical modeling for the prediction of drug synergies [47].

Note that six of the seven models contain source nodes, which are usually fixed to either 0 or 1 in most previous Boolean network analysis [26]. As targeting more comprehensive analysis of the real-world models, we did not fix source nodes, i.e., we consider all possible values of a source node. In addition, CABEAN requires applying a network reduction technique to the input model. This reduction technique fully conserves the attractors of an ABN [30]. To ensure the fairness of the evaluation, we also applied this reduction technique to all the real-world models before running them on the compared methods. Then, we ran all the benchmarks on a virtual machine whose environment is CPU: Intel(R) Core(TM) i7-3630QM 2.40GHz x 4, Memory: 8 GB, Ubuntu 18.04.2 64 bit. The time limit for each model is 10 hours.

Table 2 shows the experimental results on real-world models. Each model (except CELL CYCLE 2019) has at least one cyclic attractor, which indicates that its attraction detection does not fall to the trivial case. In general, mtsNFVS completely outperforms all the other methods. More specifically, we analyze the obtained results of each method as follows.

PyBoolNet. In all the seven models, PyBoolNet failed to compute attractors within the time limit. For each model, PyBoolNet quickly computed the approximations, but took long time for checking the correctness of the approximations via model checking. This observation is consistent with that shown in [23].

CABEAN. In all the seven models, CABEAN failed to finish the computation. We observed that in six models (except the CELL CYCLE 2019 model), CABEAN terminated before exceeding the time limit and the segmentation fault error was printed. Anyway, we also reported the time when CABEAN terminated for each model (see Table 2). Even in three models (T-LGL, CACC, CASCADE 3.0), that number is quite large. The reason for this may be that the real-world models have not only many nodes but also complex structures (see Table 1). Especially, the decomposition approach of CABEAN heavily relies on SCCs of the interaction graph, whereas the largest SCC size of each model is large. In this case, the decomposition of CABEAN may not reduce the complexity of the model enough to continue with its attractor search; leading to the termination before exceeding the time limit. This observation is consistent with that presented in [6].

AEON. This method failed to compute attractors within the time limit for the three models (SPIC, CELL CYCLE 2019, and CASCADE 3.0). It is apparent because SPIC and CASCADE 3.0 are here the two largest models with very complex structures, CELL CYCLE 2019 is here the most complex model, whereas AEON generally relies on traversing the STG although it uses some heuristics to speed up the traversal. In the four remaining models, AEON succeeded to finish the computation in reasonable time. As compared to PyBoolNet and CABEAN, AEON is more efficient. This observation is consistent with the comparison between AEON and CABEAN [6], which shows that AEON robustly outperforms CABEAN.

pystablemotifs. We first note that this method computes exact fixed points and quasi-attractors, which correspond to but may not be identical to cyclic attractors of an ABN. In some cases, it may return only a lower bound and an upper bound for the number of attractors or it may not verify completely the attractor's existence due to computational limits [36]. In the latter case, pystablemotifs manages its reactions to computation limits through the parameter *simsize*[2]. In the four models (IL-6, CELL CYCLE 2019, SIPC, and CASCADE 3.0), pystablemotifs failed to finish the computation within the time limit. For the CELL CYCLE 2019, SIPC, and CASCADE 3.0 models, the reason may be that the numbers of directed cycles in the parity-expanded networks are computationally intractable. Indeed, these three models have many nodes as well the largest SCCs of large size (see Table 1). For the IL-6 model, the reason may be that it has many source nodes, which is one of the inefficient cases of pystablemotifs [36]. For the two models (T-LGL and AD), pystablemotifs returned only intervals for the numbers of attractors ([318-336] and [2-3], respectively). We tried to increase *simsize* to 100 hoping to obtain more precise results; however, pystablemotifs obtained the run-time error after 2061.53s and 3271.44s, respectively. The observation on the T-LGL model is consistent with that shown in [35]. For the CACC model, pystablemotifs returned the exact number of attractors. However, all the cyclic attractors computed by pystablemotifs and AEON are not the same. Specifically, pystablemotifs returned two quasi-attractors of size 8, two quasi-attractors of size 128, two quasi-attractors of size 32786, and two quasi-attractors of size 262144; whereas, AEON returned two attractors of size 6, two attractors of size 96, two attractors of size 10240, and two attractors of size 61440. To sum up, pystablemotifs is unable to fully analyze any model; thus it is less efficient than AEON.

iFVS-ABN. In the five models (T-LGL, CACC, AD, IL-6, and CASCADE 3.0), iFVS-ABN succeeded to finish the computation in reasonable time. The running time of iFVS-ABN is comparable to that of AEON for the T-LGL, CACC, AD, and IL-6 models. In particular, iFVS-ABN only took 969.20s for computing all the attractors of the CASCADE 3.0 model, whereas all CABEAN, AEON, PyBoolNet, and pystablemotifs failed. Like these methods, iFVS-ABN also failed to finish the computation within the time limit for the CELL CYCLE 2019 and SIPC models, the two most complex ones. We observed that iFVS-ABN even failed to compute the candidate set F within the time limit for these models. The reason may be that the NFVS computed by iFVS-ABN is large (see Table 1) and the Boolean functions are complex, leading to too many candidate states that

[2]As the authors usually use in their benchmarks, we set this parameter as 20 in our benchmarks.

Table 1: Structure information of the real-world models. Columns 2-6 denote the number of nodes, the number of edges, the average connectivity, the number of source nodes, and the size of the largest SCC of each model, respectively. Columns 7-8 denote the sizes of the NFVSs computed by iFVS-ABN and mtsNFVS for each model, respectively.

						smallest NFVS size	
Model	# nodes	# edges	avg. con.	# source nodes	largest SCC size	iFVS-ABN	mtsNFVS
T-LGL	61	193	3.16	7	43	20	13
CACC	70	153	2.19	1	65	13	10
AD	77	206	2.68	1	72	16	11
IL-6	86	164	1.91	15	38	10	4
CELL CYCLE 2019	87	375	4.31	1	57	32	26
SIPC	133	449	3.38	11	100	29	13
CASCADE 3.0	183	603	3.30	0	176	27	19

Table 2: Timing comparisons among attractor detection methods on the selected real-world models. Columns 2-3 denote the numbers of fixed points and cyclic attractors of each model, respectively. The computational time is in seconds. "N/A" denotes a run-time error; in this case, the number inside the parentheses shows the time when encountering the error. "DNF" means that the method did not finish the attractor computation within 10 hours.

Model	# fixed points	# cyclic	CABEAN	AEON	PyBoolNet	pystablemotifs	iFVS-ABN	mtsNFVS
T-LGL	172	146	N/A (2176.55)	80.93	DNF	2267.40	547.17	8.16
CACC	2	8	N/A (21264.55)	469.38	DNF	16459.35	2898.34	1.96
AD	0	2	N/A (41.00)	5646.10	DNF	626.71	84.28	3.54
IL-6	28762	4096	N/A (19.68)	3625.36	DNF	DNF	2881.75	82.68
CELL CYCLE 2019	8	0	DNF	DNF	DNF	DNF	DNF	80.48
SIPC	640	2120	N/A (399.19)	DNF	DNF	DNF	DNF	1768.35
CASCADE 3.0	0	1	N/A (5039.31)	DNF	DNF	DNF	969.20	10.10

are unmanageable by the SAT-based approach for computing fixed points of the reduced STG [17].

mtsNFVS. Our proposed method succeeded to finish the computation within the time limit for all the seven models. The running time for each model (except SIPC) is very short (less than two minutes). In the SIPC model, the running time is longer because this model has many minimal trap spaces and in most of them mtsNFVS had to proceed Preprocessing SSF to get the optimal case (i.e., the number of candidates in the minimal trap space is one). The IL-6 model has more minimal trap spaces but in all of them the number of candidates is already one and mtsNFVS did not need to proceed Preprocessing SSF. Hence, the running time of mtsNFVS for the IL-6 model is much less than that for the SIPC model. In particular, among the considered methods only mtsNFVS can fully analyze the CELL CYCLE 2019 and SIPC models, the two most complex ones. We note that the number of attractors computed by mtsNFVS is identical to the number of minimal trap spaces for all the seven models. One can argue that using PyBoolNet [23] is also enough for these models. However, PyBoolNet cannot anticipate the case and it always needs to verify the resulting approximations. Moreover, even when the number of attractors equals the number of minimal trap spaces, the result of PyBoolNet still may be incorrect due to random walks [23]. The above experimental results confirm the accuracy advantage of mtsNFVS as compared to PyBoolNet and the time advantage of mtsNFVS as compared to iFVS-ABN (see the last paragraph of Subsection 3.1).

Finally, we report several observations on the constituent tasks of mtsNFVS. Regarding the computation of an NFVS, as compared to iFVS-ABN, mtsNFVS computed smaller NFVSs in all the seven models (see Table 1); leading to smaller candidate sets. This observation is evident for the usefulness of the algorithm [11] that we use to compute an NFVS of the ABN (see Subsection 3.2). Regarding the computation of the set of fixed points of the reduced STG and the set of minimal trap spaces, mtsNFVS finished the computation within very short time (less than one minute for all models except SIPC). For the SIPC model, the computational time for computing fixed points (resp. minimal trap spaces) is a bit longer (901.64s (resp. 238.05s)) because there are many fixed points (resp. minimal trap spaces) as well as its Boolean functions are complex. This observation is evident for the usefulness of the ASP-based methods that we use in mtsNFVS (see Subsections 3.3 and 3.2, respectively). Regarding Preprocessing SSF, for all the seven models it always lead to the best cases where mtsNFVS did not need to perform the reachability analysis. In particular, the computational time of Preprocessing SSF is very short (less than one minute for all models except SIPC). For the SIPC model, the computational time for Preprocessing SSF is a bit longer because the number of candidates is initially large ($|F| = 24800$). For the IL-6 model, the number of candidates is larger ($|F| = 32768$). However, that number equals the number of minimal trap spaces and mtsNFVS did not need to perform Preprocessing SSF. Hence, the running time for the attractor computation of the IL-6 model is much less than that of the SIPC model. This observation is evident for the usefulness of Preprocessing SSF.

4.2 Experimental results on random models

We randomly generated a set of models by using Bool Net R package [29]. This set includes N-K models [22] with network size $n \in \{100, 150, 200, 250\}$ and $K = 2$ (i.e., each node has exactly $K = 2$ input nodes). For each network size, 10 instances were generated using the *generateRandomNKNetwork* function. In total, we have 40 random N-K models. We then applied the compared methods to these models and recorded the number of failures (i.e., failed to obtain the result within three hours) of each method. Other settings are the same as those in Subsection 4.1.

Table 3: Number of failures on N-K models.

Method	n = 100	n = 150	n = 200	n = 250
CABEAN	10	10	10	10
AEON	2	10	10	10
PyBoolNet	10	10	10	10
pystablemotifs	9	9	10	10
iFVS-ABN	1	6	9	10
mtsNFVS	0	0	0	0

Table 3 shows the experimental results on N-K models. From these results, we obtained several observations similar to those obtained in Subsection 4.1. More specifically, CABEAN and PyBoolNet failed to compute attractors within the time limit for all the models. AEON and pystablemotif can handle only a few models for $n = 100$ and their numbers of failures rapidly approach 10 for larger n. Although iFVS-ABN is better, its number of failures drastically increases. For $n = 200$, iFVS-ABN can handle only one model, which is consistent with the observation presented in [17]. In the case of mtsNFVS, it can handle all the models. Furthermore, the number of failures of mtsNFVS is always less than that of other methods in each network size. We also reported that the running time of mtsNFVS in each model is always less than 800 seconds. These observations show that mtsNFVS completely outperforms all the other methods and it can handle large-scale models.

5 CONCLUSION AND FUTURE WORK

In this paper, we have proposed the novel method mtsNFVS for computing all the attractors of an ABN. mtsNFVS exploits the advantages of the efficient method for computing minimal trap spaces of the ABN and the NFVS-based approach of iFVS-ABN. In principle, similar to iFVS-ABN, mtsNFVS also relies on NFVSs and the reducing dynamics to get a candidate set of states, then filters out this set to compute attractors of the ABN. However, by using minimal trap spaces mtsNFVS can open a chance to reach easy cases for the reachability analysis in the filtering process, which are generally unable in iFVS-ABN. To make the chance effective, we have then proposed several algorithmic improvements to several common constituent tasks between mtsNFVS and iFVS-ABN. These improvements along with the use of minimal trap spaces are key factors for the efficiency of mtsNFVS.

We tested the method on large and complex real-world Boolean models as well as randomly generated models. The experimental results show that mtsNFVS completely outperforms the state-of-the-art method CABEAN and other previously notable methods. In particular, mtsNFVS can handle the two most complex models of the real-world model set, whereas all other methods cannot. Since the analysis for these models was usually performed by using their reduced versions due to the performance limitations of the existing tools, the efficiency of mtsNFVS pushes the barrier of what was previously possible in the field of systems biology.

In the future, we plan to conduct a more comprehensive comparison among the existing methods for attractor detection in ABNs (i.e., CABEAN, AEON, PyBoolNet, pystablemotifs, iFVS-ABN, and mtsNFVS). To do this, we shall need to run experiments on more real-world models as well as randomly generated models of larger size. In addition, the processes of mtsNFVS for the set of minimal trap spaces seem potentially to be paralleled. Hence, we plan to investigate deeply the parallelization capability of mtsNFVS. Finally, proposing heuristics to help Preprocessing SSF converge more quickly is also a potential improvement to mtsNFVS.

REFERENCES

[1] Tatsuya Akutsu. 2018. *Algorithms for analysis, inference, and control of Boolean networks.* World Scientific, Singapore.

[2] Tatsuya Akutsu, Morihiro Hayashida, and Takeyuki Tamura. 2009. Integer programming-based methods for attractor detection and control of Boolean networks. In *Proceedings of the 48th IEEE Conference on Decision and Control, CDC 2009.* IEEE. https://doi.org/10.1109/CDC.2009.5400017

[3] Reka Albert and Juilee Thakar. 2014. Boolean modeling: a logic-based dynamic approach for understanding signaling and regulatory networks and for making useful predictions. *Wiley Interdisciplinary Reviews: Systems Biology and Medicine* 6, 5 (2014), 353–369. https://doi.org/10.1002/wsbm.1273

[4] Jonas Béal, Lorenzo Pantolini, Vincent Noël, Emmanuel Barillot, and Laurence Calzone. 2021. Personalized logical models to investigate cancer response to BRAF treatments in melanomas and colorectal cancers. *PLoS Comput. Biol.* 17, 1 (2021). https://doi.org/10.1371/journal.pcbi.1007900

[5] Nikola Benes, Lubos Brim, Jakub Kadlecaj, Samuel Pastva, and David Safránek. 2020. AEON: Attractor Bifurcation Analysis of Parametrised Boolean Networks. In *Computer Aided Verification - 32nd International Conference, CAV 2020, Los Angeles, CA, USA, July 21-24, 2020.* Springer, 569–581. https://doi.org/10.1007/978-3-030-53288-8_28

[6] Nikola Benes, Lubos Brim, Samuel Pastva, and David Safránek. 2021. Computing Bottom SCCs Symbolically Using Transition Guided Reduction. In *Computer Aided Verification - 33rd International Conference, CAV 2021, Virtual Event, July 20-23, 2021.* Springer, 505–528. https://doi.org/10.1007/978-3-030-81685-8_24

[7] Célia Biane and Franck Delaplace. 2019. Causal Reasoning on Boolean Control Networks Based on Abduction: Theory and Application to Cancer Drug Discovery. *IEEE ACM Trans. Comput. Biol. Bioinform.* 16, 5 (2019), 1574–1585. https://doi.org/10.1109/TCBB.2018.2889102

[8] Claudine Chaouiya, Aurélien Naldi, and Denis Thieffry. 2011. Logical Modelling of Gene Regulatory Networks with GINsim. In *Bacterial Molecular Networks.* Springer New York, 463–479. https://doi.org/10.1007/978-1-61779-361-5_23

[9] Thomas Chatain, Stefan Haar, Juraj Kolcák, Loïc Paulevé, and Aalok Thakkar. 2020. Concurrency in Boolean networks. *Nat. Comput.* 19, 1 (2020), 91–109. https://doi.org/10.1007/s11047-019-09748-4

[10] Leonardo De Moura and Nikolaj Bjørner. 2008. Z3: An efficient SMT solver. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems.* Springer, 337–340. https://doi.org/10.1007/978-3-540-78800-3_24

[11] Philippe Galinier, Eunice Lemamou, and Mohamed Wassim Bouzidi. 2013. Applying local search to the feedback vertex set problem. *Journal of Heuristics* 19, 5 (June 2013), 797–818. https://doi.org/10.1007/s10732-013-9224-z

[12] Abhishek Garg, Alessandro Di Cara, Ioannis Xenarios, Luis Mendoza, and Giovanni De Micheli. 2008. Synchronous versus asynchronous modeling of gene regulatory networks. *Bioinform.* 24, 17 (2008), 1917–1925. https://doi.org/10.1093/bioinformatics/btn336

[13] Abhishek Garg, Ioannis Xenarios, Luis Mendoza, and Giovanni De Micheli. 2007. An Efficient Method for Dynamic Analysis of Gene Regulatory Networks and *in silico* Gene Perturbation Experiments. In *Research in Computational Molecular Biology, 11th Annual International Conference, RECOMB 2007, Oakland, CA, USA, April 21-25, 2007.* Springer, 62–76. https://doi.org/10.1007/978-3-540-71681-5_5

[14] Carlos Gershenson. 2004. Introduction to random Boolean networks. In *Proceedings of the Ninth International Conference on the Simulation and Synthesis of Living Systems (ALife IX).* MIT Press, 160–173.

[15] Trinh Van Giang and Kunihiko Hiraishi. 2020. An efficient method for approximating attractors in large-scale asynchronous Boolean models. In *IEEE International*

Conference on Bioinformatics and Biomedicine, BIBM 2020, Virtual Event, South Korea, December 16-19, 2020. IEEE, 1820–1826. https://doi.org/10.1109/BIBM49941.2020.9313230

[16] Trinh Van Giang and Kunihiko Hiraishi. 2020. A Study on Attractors of Generalized Asynchronous Random Boolean Networks. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.* 103-A, 8 (2020), 987–994. https://doi.org/10.1587/transfun.2019EAP1163

[17] Trinh Van Giang and Kunihiko Hiraishi. 2021. An Improved Method for Finding Attractors of Large-Scale Asynchronous Boolean Networks. In *IEEE Conference on Computational Intelligence in Bioinformatics and Computational Biology, CIBCB 2021, Melbourne, Australia, October 13-15, 2021*. IEEE, 1–9. https://doi.org/10.1109/CIBCB49929.2021.9562947

[18] Inman Harvey and Terry Bossomaier. 1997. Time out of joint: Attractors in asynchronous random Boolean networks. In *Proceedings of the Fourth European Conference on Artificial Life*. Citeseer, 67–75.

[19] Sui Huang. 2001. Genomics, complexity and drug discovery: insights from Boolean network models of cellular regulation. *Pharmacogenomics* 2, 3 (2001), 203–222. https://doi.org/10.1517/14622416.2.3.203

[20] Itziar Irurzun-Arana, José Martín Pastor, Iñaki F. Trocóniz, and José David Gómez-Mantilla. 2017. Advanced Boolean modeling of biological networks applied to systems pharmacology. *Bioinform.* 33, 7 (2017), 1040–1048. https://doi.org/10.1093/bioinformatics/btw747

[21] David S Johnson and Michael R Garey. 1979. *Computers and intractability: A guide to the theory of NP-completeness*. W. H. Freeman, New York.

[22] S.A. Kauffman. 1969. Metabolic stability and epigenesis in randomly constructed genetic nets. *Journal of Theoretical Biology* 22, 3 (March 1969), 437–467. https://doi.org/10.1016/0022-5193(69)90015-0

[23] Hannes Klarner and Heike Siebert. 2015. Approximating Attractors of Boolean Networks by Iterative CTL Model Checking. *Frontiers in Bioengineering and Biotechnology* 3 (Sept. 2015). https://doi.org/10.3389/fbioe.2015.00130

[24] Hannes Klarner, Adam Streck, and Heike Siebert. 2017. PyBoolNet: a python package for the generation, analysis and visualization of Boolean networks. *Bioinform.* 33, 5 (2017), 770–772. https://doi.org/10.1093/bioinformatics/btw682

[25] Junyan Lu, Hanlin Zeng, Zhongjie Liang, Limin Chen, Liyi Zhang, Hao Zhang, Hong Liu, Hualiang Jiang, Bairong Shen, Ming Huang, Meiyu Geng, Sarah Spiegel, and Cheng Luo. 2015. Network modelling reveals the mechanism underlying colitis-associated colon cancer and identifies novel combinatorial anti-cancer targets. *Scientific Reports* 5, 1 (Oct. 2015). https://doi.org/10.1038/srep14739

[26] Andrzej Mizera, Jun Pang, Hongyang Qu, and Qixia Yuan. 2019. Taming Asynchrony for Attractor Detection in Large Boolean Networks. *IEEE ACM Trans. Comput. Biol. Bioinform.* 16, 1 (2019), 31–42. https://doi.org/10.1109/TCBB.2018.2850901

[27] Arnau Montagud, Jonas Béal, Luis Tobalina, Pauline Traynard, Vigneshwari Subramanian, Bence Szalai, Róbert Alföldi, László Puskás, Alfonso Valencia, Emmanuel Barillot, Julio Saez-Rodriguez, and Laurence Calzone. 2021. Patient-specific Boolean models of signaling networks guide personalized treatments. (July 2021). https://doi.org/10.1101/2021.07.28.454126

[28] Marco Montalva, Julio Aracena, and Anahí Gajardo. 2008. On the complexity of feedback set problems in signed digraphs. *Electron. Notes Discrete Math.* 30 (2008), 249–254. https://doi.org/10.1016/j.endm.2008.01.043

[29] Christoph Müssel, Martin Hopfensitz, and Hans A Kestler. 2010. BoolNet—an R package for generation, reconstruction and analysis of Boolean networks. *Bioinformatics* 26, 10 (2010), 1378–1380.

[30] Aurélien Naldi, Pedro T. Monteiro, and Claudine Chaouiya. 2012. Efficient Handling of Large Signalling-Regulatory Networks by Focusing on Their Core Control. In *Computational Methods in Systems Biology - 10th International Conference, CMSB 2012, London, UK, October 3-5, 2012*. Springer, 288–306. https://doi.org/10.1007/978-3-642-33636-2_17

[31] Vincent Noël, Jose Carbonell, Miguel Ponce de Leon, Sylvain Soliman, Anna Niarakis, Laurence Calzone, Emmanuel Barillot, Alfonso Valencia, and Arnau Montagud. 2020. PhysiBoSS-COVID: the Boolean modelling of COVID-19 signalling pathways in a multicellular simulation framework allows for the uncovering of mechanistic insights. https://doi.org/10.5281/zenodo.4266778

[32] Oyebode J. Oyeyemi, Oluwafemi Davies, David L. Robertson, and Jean-Marc Schwartz. 2015. A logical model of HIV-1 interactions with the T-cell activation signalling pathway. *Bioinform.* 31, 7 (2015), 1075–1083. https://doi.org/10.1093/bioinformatics/btu787

[33] Jong-Chan Park, So-Yeong Jang, Dongjoon Lee, Jeongha Lee, Uiryong Kang, Hongjun Chang, Haeng Jun Kim, Sun-Ho Han, Jinsoo Seo, Murim Choi, Dong Young Lee, Min Soo Byun, Dahyun Yi, Kwang-Hyun Cho, and Inhee Mook-Jung. 2021. A logical network-based drug-screening platform for Alzheimer's disease representing pathological features of human brain organoids. *Nature Communications* 12, 1 (Jan. 2021). https://doi.org/10.1038/s41467-020-20440-5

[34] Loïc Paulevé and Adrien Richard. 2012. Static analysis of Boolean networks based on interaction graphs: A survey. *Electron. Notes Theor. Comput. Sci.* 284 (2012), 93–104. https://doi.org/10.1016/j.entcs.2012.05.017

[35] Jordan C Rozum, Dávid Deritei, Kyu Hyong Park, Jorge Gómez Tejeda Zañudo, and Réka Albert. 2021. pystablemotifs: Python library for attractor identification and control in Boolean networks. *Bioinformatics* (2021). https://doi.org/10.1093/bioinformatics/btab825

[36] Jordan C. Rozum, Jorge Gómez Tejeda Zañudo, Xiao Gan, Dávid Deritei, and Réka Albert. 2021. Parity and time reversal elucidate both decision-making in empirical models and attractor scaling in critical Boolean networks. *Science Advances* 7, 29 (July 2021). https://doi.org/10.1126/sciadv.abf8124

[37] Anke Ryll, Regina Samaga, Fred Schaper, Leonidas G. Alexopoulos, and Steffen Klamt. 2011. Large-scale network models of IL-1 and IL-6 signalling and their hepatocellular specification. *Molecular BioSystems* 7, 12 (2011), 3253. https://doi.org/10.1039/C1MB05261F

[38] Assieh Saadatpour, István Albert, and Réka Albert. 2010. Attractor analysis of asynchronous Boolean models of signal transduction networks. *Journal of Theoretical Biology* 266, 4 (Oct. 2010), 641–656. https://doi.org/10.1016/j.jtbi.2010.07.022

[39] Julian D Schwab, Silke D Kühlwein, Nensi Ikonomi, Michael Kühl, and Hans A Kestler. 2020. Concepts in Boolean network modeling: What do they all mean? *Computational and Structural Biotechnology Journal* 18 (2020), 571–582. https://doi.org/10.1016/j.csbj.2020.03.001

[40] Shubhank Sherekar and Ganesh A Viswanathan. 2021. Boolean dynamic modeling of cancer signaling networks: Prognosis, progression, and therapeutics. *Computational and Systems Oncology* 1, 2 (2021), e1017. https://doi.org/10.1002/cso2.1017

[41] Herbert Sizek, Andrew Hamel, Dávid Deritei, Sarah Campbell, and Erzsébet Ravasz Regan. 2019. Boolean model of growth signaling, cell cycle and apoptosis predicts the molecular mechanism of aberrant cell cycle progression driven by hyperactive PI3K. *PLOS Computational Biology* 15, 3 (March 2019), e1006402. https://doi.org/10.1371/journal.pcbi.1006402

[42] Thomas Skodawessely and Konstantin Klemm. 2011. Finding attractors in Asynchronous Boolean Dynamics. *Adv. Complex Syst.* 14, 3 (2011), 439–449. https://doi.org/10.1142/S0219525911003098

[43] Cui Su and Jun Pang. 2021. Cabean 2.0: Efficient and Efficacious Control of Asynchronous Boolean Networks. In *Formal Methods - 24th International Symposium, FM 2021, Virtual Event, November 20-26, 2021*. Springer, 581–598. https://doi.org/10.1007/978-3-030-90870-6_31

[44] Cui Su, Jun Pang, and Soumya Paul. 2021. Towards Optimal Decomposition of Boolean Networks. *IEEE ACM Trans. Comput. Biol. Bioinform.* 18, 6 (2021), 2167–2176. https://doi.org/10.1109/TCBB.2019.2914051

[45] René Thomas. 1973. Boolean formalization of genetic control circuits. *Journal of Theoretical Biology* 42, 3 (Dec. 1973), 563–585. https://doi.org/10.1016/0022-5193(73)90247-6

[46] G. V. Trinh, T. Akutsu, and K. Hiraishi. 2020. An FVS-based Approach to Attractor Detection in Asynchronous Random Boolean Networks. *IEEE/ACM Trans. Comput. Biol. Bioinf.* (2020). https://doi.org/10.1109/TCBB.2020.3028862 in press.

[47] Eirini Tsirvouli, Vasundra Touré, Barbara Niederdorfer, Miguel Vázquez, Åsmund Flobak, and Martin Kuiper. 2020. A Middle-Out Modeling Strategy to Extend a Colon Cancer Logical Model Improves Drug Synergy Predictions in Epithelial-Derived Cancer Cell Lines. *Frontiers in Molecular Biosciences* 7 (Oct. 2020). https://doi.org/10.3389/fmolb.2020.502573

[48] Arash Vahidi. 2003. JDD: a pure Java BDD and Z-BDD library. https://bitbucket.org/vahidi/jdd.

[49] José C. Valverde, Henning S. Mortveit, Carlos Gershenson, and Yongtang Shi. 2020. Boolean Networks and Their Applications in Science and Engineering. *Complex.* 2020 (2020), 6183798:1–6183798:3. https://doi.org/10.1155/2020/6183798

[50] Alan Veliz-Cuba, Boris Aguilar, Franziska Hinkelmann, and Reinhard Laubenbacher. 2014. Steady state analysis of Boolean molecular network models via model reduction and computational algebra. *BMC Bioinformatics* 15, 1 (June 2014). https://doi.org/10.1186/1471-2105-15-221

[51] Jorge GT Zañudo and Réka Albert. 2013. An effective network reduction approach to find the dynamical repertoire of discrete dynamic networks. *Chaos: An Interdisciplinary Journal of Nonlinear Science* 23, 2 (2013), 025111. https://doi.org/10.1063/1.4809777

[52] R. Zhang, M. V. Shah, J. Yang, S. B. Nyland, X. Liu, J. K. Yun, R. Albert, and T. P. Loughran. 2008. Network model of survival signaling in large granular lymphocyte leukemia. *Proceedings of the National Academy of Sciences* 105, 42 (Oct. 2008), 16308–16313. https://doi.org/10.1073/pnas.0806447105

[53] Desheng Zheng, Guowu Yang, Xiaoyu Li, Zhicai Wang, Feng Liu, and Lei He. 2013. An efficient algorithm for computing attractors of synchronous and asynchronous Boolean networks. *PloS One* 8, 4 (2013), e60593. https://doi.org/10.1371/journal.pone.0060593