SCIENCE CHINA
Information Sciences

• **RESEARCH PAPER** •

Improving BDD-based Attractor Detection for Synchronous Boolean Networks

Qixia YUAN[1]*, Hongyang QU[2], Jun PANG[1,3] & Andrzej MIZERA[1]

[1]*Faculty of Science, Technology and Communication, University of Luxembourg, Luxembourg;*
[2]*Department of Automatic Control & Systems Engineering, University of Sheffield, UK;*
[3]*Interdisciplinary Centre for Security, Reliability and Trust, University of Luxembourg, Luxembourg*

Received January 1, 2016; accepted January 1, 2016; published online January 1, 2016

Abstract Boolean networks are an important formalism for modelling biological systems and have attracted much attention in recent years. An important challenge in Boolean networks is to exhaustively find attractors, which represent steady states of a biological network. In this paper, we propose a new approach to improve the efficiency of BDD-based attractor detection. Our approach includes a monolithic algorithm for small networks, an enumerative strategy to deal with large networks, a method to accelerate attractor detection based on an analysis of the network structure, and two heuristics on ordering BDD variables. We demonstrate the performance of our approach on a number of examples and on a realistic model of apoptosis in hepatocytes. We compare it with one existing technique in the literature.

Keywords Boolean networks, systems biology, binary decision diagram, attractor, verification algorithms

Citation Yuan Q, Qu H, Pang J, et al. Improving BDD-based Attractor Detection for Synchronous Boolean Networks. Sci China Inf Sci, 2016, 59(1): xxxxxx, doi: xxxxxxxxxxxxxx

1 Introduction

A gene regulatory network (GRN) is a collection of genes, proteins and other regulatory molecules that interact with each other indirectly, govern the expression of genes and ultimately regulate the cellular behaviours. Acquiring insights into the dynamics of GRNs has attracted much attention of researchers. In particular, *attractors* of a GRN are of special interest. They are hypothesised to characterise cellular phenotypes [12] or to correspond to functional cellular states such as proliferation, apoptosis, or differentiation [10]. In this paper, we focus on detecting attractors for GRNs modelled with Boolean networks (BNs) [12, 21].

In the BN framework, algorithms for detecting attractors have been extensively studied in the literature. One important type of algorithms among them is based on the Binary Decision Diagram (BDD). In this paper, we concentrate on improving the BDD-based attractor detection algorithm for synchronous BNs demonstrated in [8]. The algorithm works well for small BNs, but becomes inefficient for large BNs. We improve this algorithm with regard to three aspects. Firstly, we propose to use different attractor detection strategies for networks of different sizes. For small BNs, we propose to use a monolithic

* Corresponding author (email: qixia.yuan@uni.lu)

algorithm which searches for attractors starting from the whole state space, instead of one initial state, in order to reduce the time for attractor detection. For large BNs, we improve the algorithm in [8] by computing the predecessor states of each individual state in a detected attractor, instead of computing the predecessors of the whole attractor, when pruning the searching space for the subsequent detection of the remaining attractors. Our second contribution lies in accelerating the attractor detection of large networks. We design a decomposition algorithm to reduce the state space to be searched. Lastly, we propose two heuristics on ordering BDD variables based on the structure of BNs to further improve the attractor detection algorithm. Experiments show that our proposed algorithms can effectively reduce the time cost for BDD-based attractor detection and outputform the existing tool GenYsis [8] for more than 100 times in some cases.

Structure of the paper. The rest of the paper is organised as follows. We discuss the related work in Section 2. After presenting preliminaries on Boolean networks and BDD in Section 3, we introduce our proposed attractor detection algorithms in Section 4 and two methods for accelerating the attractor detection algorithms in Section 5. We develop two BDD variables order heuristics in Section 6 and present evaluation results in Section 7. We discuss some future work in Section 9.

2 Related Work

In this section we review a number of different approaches for attractor detection in the framework of BNs. The simplest way to detect attractors is to enumerate all the possible states and to run simulation from each one until an attractor is reached [24]. This method ensures that all the attractors are detected but it has exponential time complexity and its applicability is highly restricted by the network size. Another approach is to take a sample from the whole state space and simulate from it until an attractor is found [16]. However, this technique cannot guarantee finding all the attractors of a BN. Later, Irons proposed a method by analysing partial states involving parts of the nodes [11]. This method can reduce the computational complexity of attractor detection from exponential time to polynomial time; however, it is highly dependent on the topology of the underlying network and the network size manageable by this method is restricted to 50.

Next, the efficiency and scalability of attractor detection techniques are further improved with the integration of two techniques. This first technique is based on Binary Decision Diagram (BDD), a compact data structure for representing Boolean functions. Algorithms proposed in [6–8] explore BDDs to encode the Boolean functions in BNs, use BDD operations to capture the dynamics of the networks, and to build their corresponding transition systems. The efficient operations of BDDs are used to compute the forward and backward reachable states. Attractor detection is then reduced to finding self-loops or simple cycles in the transition systems, which highly relies on the computation of forward and backward reachable states. Garg *et al.* proposed a method for detecting attractors for both synchronous and asynchronous BNs [8]. Later in [7], the method was further improved for attractor detection of asynchronous BNs. In a recent work [27], Zheng *et al.* developed an algorithm based on reduced-order BDD (ROBDD) data structure, which further speeds up the computation time of attractor detection. These BDD-based solutions only work for GRNs of a hundred of nodes and suffer from the infamous state explosion problem, as the size of the BDD depends both on the regulatory functions and the number of nodes in the GRNs.

The other technique represents attractor detection in BNs as a satisfiability (SAT) problem [5]. The main idea is inspired by SAT-based bounded model checking: the transition relation of the GRN is unfolded into a bounded number of steps in order to construct a propositional formula which encodes attractors and which is then solved by a SAT solver. In every unfolding step a new variable is required to represent a state of a node in the GRN. It is clear that the efficiency of these algorithms largely depends on the number of unfolding steps required and the number of nodes in the GRN.

Recently, decomposition based algorithms have been developed for dealing with large BNs. Zhao *et at.* proposed an aggregation algorithm to deal with large BNs. Their idea is to decompose a large BN into several sub-networks and detect attractors of each sub-network [26]. By merging the attractors

of all the sub-networks, their algorithm can reveal the attractors of the original BN. In [9], Guo *et al.* developed an SCC (strongly connected component)-based decomposition method. Their method divides a BN into several sub-networks according to the SCCs in the BN and assigns each sub-network a credit. The attractor detection in each sub-network is performed one by one according to their credits. Unlike the algorithm in [26], when detecting attractors of a sub-network, the method of Guo *et al.* considers the attractor information of other sub-networks whose credits are smaller. In this way, it reveals the attractors of the original BN by detecting attractors of the last sub-network. In our work, we are inspired by the ideas in [9, 26] to explore the structure of large BNs to accelerate the attractor detection process. Our main idea is to use all attractor information of small sub-networks, to which efficient attractor detection algorithms for small BNs can be applied, to reduce the set of initial states of the original BN and thus speed up the attractor detection of the original but large network.

3 Preliminaries

3.1 Boolean networks

Boolean networks (BNs) are a class of discrete dynamical systems in which interactions over a set of Boolean variables are considered. They were first introduced by Stuart Kauffman in 1969 as a class of simple models for the analysis of the dynamical properties of gene regulatory networks [13], where projection of gene states to an ON/OFF pattern of binary states was considered. Formally, a *Boolean Network* $G(V, \boldsymbol{f})$ is defined as a set of binary-valued variables, also referred to as nodes or genes, $V = \{x_1, x_2, \ldots, x_n\}$ and a vector of Boolean functions $\boldsymbol{f} = (f_1, f_2, \ldots, f_n)$. The time is discrete and in fact the consecutive time points correspond to switching events of states of the genes. At each time point t ($t = 0, 1, \ldots$), the *state* of the network is defined by the vector $\boldsymbol{x}(t) = (x_1(t), x_2(t), \ldots, x_n(t))$, where $x_i(t)$ is the value of variable x_i at time t, i.e., $x_i(t) \in \{0, 1\}$ ($i = 1, 2, \ldots, n$).

For each variable x_i, there exists a *predictor set* (parent nodes set) $\{x_{i_1}, x_{i_2}, \ldots, x_{i_{k(i)}}\}$, and a Boolean *predictor function* (or simply *predictor*) f_i being the i-th element of $\boldsymbol{f}$ that determines the value of x_i at the next time point, i.e., $x_i(t+1) = f_i(x_{i_1}(t), x_{i_2}(t), \ldots, x_{i_{k(i)}}(t))$, where $1 \leqslant i_1 < i_2 < \cdots < i_{k(i)} \leqslant n$. Since the predictor functions of $\boldsymbol{f}$ are time-homogenous, the notation can be simplified by writing $f_i(x_{i_1}, x_{i_2}, \ldots, x_{i_{k(i)}})$. A node x_{i_j} with $1 \leqslant j \leqslant k(i)$ in the parent nodes set is called a parent node of x_i and x_i is called a child node of x_{i_j}. The parent node set can be further divided into two sets, i.e., the *activator set* which contains genes that can activate x_i and the *inhibitor set* which contains genes that can inhibit x_i. In general, the Boolean functions can be formed by combinations of any logical operators. However, in the context of GRNs usually the set of possible Boolean functions is restricted (see e.g., [7, 20]). Here the following three types of Boolean functions are considered in GenYsis [8]: 1) $x_i(t+1) = \left(\bigvee_{j=1}^{m} x_j^a(t)\right) \wedge \neg \left(\bigvee_{j=1}^{n} x_j^{in}(t)\right)$, 2) $x_i(t+1) = \left(\bigvee_{j=1}^{m} x_j^a(t)\right)$, 3) $x_i(t+1) = \neg \left(\bigvee_{j=1}^{n} x_j^{in}(t)\right)$, where $x_j \in \{0, 1\}$; x_m^a and x_n^{in} are the set of activators and inhibitors of x_i; $\wedge$, $\vee$, and $\neg$ represent logical AND, OR, and NEGATION, respectively. The first function is used when there are both activators and inhibitors for a gene x_i. The second function is used when gene x_i only has activators as its parent nodes and the third function is used in the case where gene x_i only has inhibitors as its parent nodes. The BNs are divided into two types based on the time evolution of its states, i.e., *synchronous* and *asynchronous*. In synchronous BNs, the values of all the variables are updated simultaneously; while in asynchronous BNs, one variable is updated at a time. In this paper, we focus on synchronous BNs. The vector $\boldsymbol{f}$ of predictor functions determines the time evolution of the state of a synchronous BN, i.e.,

$$\boldsymbol{x}(t+1) = \boldsymbol{f}(\boldsymbol{x}(t)). \tag{1}$$

Thus, the BN's dynamics is fully deterministic. The only potential uncertainty lies in the selection of the initial starting state of the network.

Starting from an initial state, after a finite number of steps, a Boolean network will reach an *attractor*: either a fixed state or a set of states through which it will repeatedly cycle forever. Each fixed state is

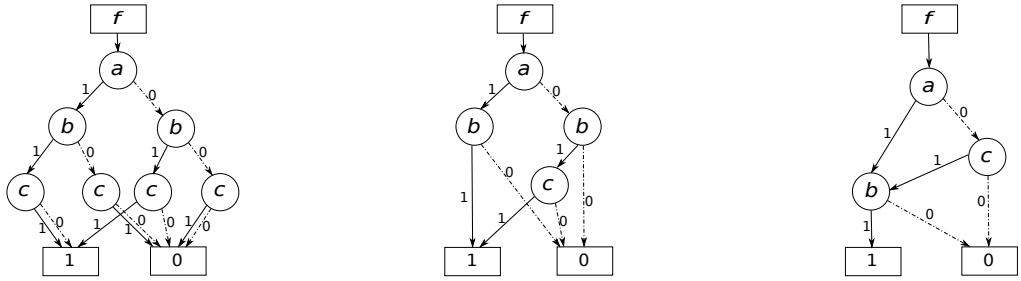

Figure 1 A BDD representing $f = (a \text{ OR } c) \text{ AND } b$. Left: full BDD; middle: reduced BDD; right: ordered reduced BDD.

referred to as a *singleton attractor* and each cyclicly revisited set of states is called a *cyclic attractor*. The number of states in a cyclic attractor is the *cycle length* of that attractor. The *attractor structure* of a BN is given by all its singleton and cyclic attractors together with their cycle lengths. The states constituting an attractor are called *attractor states*. The remaining states are called *transient* and are visited at most once on any network trajectory. An attractor and all the transient states that lead into it constitute its *basin of attraction*. In consequence, all basins of attraction of a BN form a partition of its state space. The attractor structure of a Boolean network characterises its long-run behaviour [23], and is of particular interest as, for instance, attractors are hypothesised to characterise cellular phenotypes [12].

The density of a BN is measured with its parent nodes number. For a BN G, its density is defined as $\mathcal{D}(G) = \frac{1}{n}\sum_{i=1}^{n} k(i)$, where $k(i)$ is the number of parent nodes of the i-th predictor function.

3.2 Binary Decision Diagram

A Binary Decision Diagram (BDD) is a directed acyclic graph (DAG) used to represent a Boolean function. It was originally introduced by Lee in [14] and Akers in [1]. It consists of a root node, (intermediate) decision nodes, and two terminal nodes, i.e., 0-terminal and 1-terminal. Each variable of a Boolean function is represented as a decision node in the BDD. Each decision node contains two outgoing edges, representing the two possible evaluations 0 and 1 of the variable. A path from the root node to the 1-terminal represents variable values that make the Boolean function evaluate to true; while a path from the root node to the 0-terminal represents variable values that make the Boolean function take value false. For example, a BDD that represents the Boolean function $f = (a \text{ OR } c) \text{ AND } b$ is shown in Figure 1 (left). In this figure, root node and terminal nodes are represented with squares while decision nodes are represented with circles. A solid arrow starting from a decision node represents the assignment of value 1 to that node while a dashed arrow starting from a decision node represents the assignment of value 0 for that node. Each path from the root node to one of the terminal nodes represents an assignment of the variables, e.g., the path $f \to a \xrightarrow{1} b \xrightarrow{1} c \xrightarrow{1} 1$ represents the assignment $a = 1,\ b = 1,\ c = 1$ for which the Boolean function takes the value true. If the value of a variable in a certain evaluation does not affect the value of the Boolean function, the variable can be simply removed from the path. After removing all unnecessary variables from the above BDD, we get a reduced BDD, as shown in Figure 1 (middle). The number of decision nodes in the BDD is reduced from 7 to 4. There are two paths leading to the 1-terminal node. The first path $f \to a \xrightarrow{1} b \xrightarrow{1} 1$ represents two possible assignments, i.e., $a = 1,\ b = 1,\ c = 1$ or $a = 1,\ b = 1,\ c = 0$; and the second path $f \to a \xrightarrow{0} b \xrightarrow{1} c \xrightarrow{1} 1$ represents the assignment $a = 0,\ b = 1,\ c = 1$.

Different orders of BDD variables will result in different number of decision nodes in the reduced BDD representation. For example, if we represent the above Boolean function in the order of $a \to c \to b$, as shown in Figure 1 (right), we can reduce the number of decision nodes from 4 to 3. This reduction of decision nodes directly affects the size of the BDD representation and the time cost of BDD operations. To find an optimal variable ordering for constructing a minimum-size BDD is known to be an NP-complete problem [2]. In consequence, using heuristics to construct a near optimal reduced ordered BDD is crucial for achieving good performance in BDD-based operations.

3.3 Encoding Boolean networks in BDDs

Boolean networks can be easily modelled as *transition systems*, which then can be encoded in BDDs. A transition system TS is a tuple $\langle S, S_0, T\rangle$ where 1) S is a finite set of states, 2) $S_0 \subseteq S$ is the set of initial states, and 3) $T \subseteq S \times S$ is the *transition relation*, which specifies how the system evolves in the state space. In a transition $(s_1, s_2) \in T$, state s_1 is called the *source state* and s_2 is the *target state*.

Given a Boolean network $G(V, \boldsymbol{f})$, a transition system $TS = \langle S, S_0, T\rangle$ can be generated as follows. 1) Each state of the network is a state in S. Thus, S contains 2^n states. 2) Each state of the network is an initial state in TS. Hence, in the transition system we have $S_0 = S$. 3) Each time evolution of the states of the network is a transition in T. That is, Equation (1) can be translated into a transition $(s_1, s_2) \in T$, where $s_1 = \boldsymbol{x}(t)$ and $s_2 = \boldsymbol{x}(t+1)$. As the network is deterministic, each state in TS has only one outgoing transition and one successor state.

A Boolean network $G(V, \boldsymbol{f})$ can be encoded in BDDs in order to be stored efficiently for further processing. Each variable in V can be represented by a binary BDD variable. By slightly abusing the notation, we use V to denote the set of BDD variables. In order to encode the transition relation, another set V' of BDD variables, which is a copy of V, is needed, so that V encodes the source states and V' encodes the target states. Hence, the transition relation can be seen as a function $T : V \to V'$. Our attractor detection algorithms also need two functions as the basis. 1) $Image(X, T) = \{s \in S \mid \exists s' \in X \text{ such that } (s', s) \in T\}$ returns the set of target states that can be reached from a state in $X \subseteq S$ following a transition in T. 2) $Preimage(Y, T) = \{s \in S \mid \exists s' \in Y \text{ such that } (s, s') \in T\}$ returns the set of source states that can reach a state in $Y \subseteq S$ following a transition in T.

4 Attractor Detection Algorithms

We propose in Section 4.1 a monolithic attractor detection algorithm (Algorithm 1) for dealing with small networks and an enumerative search algorithm (Algorithm 5) for large networks in Section 4.2. Algorithms 2 to 4 introduced in Section 4.2 are used by Algorithm 5. Based on Algorithms 1 and 5, we further propose a decomposition approach to accelerate the detection process in Section 5.

4.1 Monolithic attractor detection algorithm

The first algorithm is called *monolithic* algorithm. The idea is to apply the transition relation to a set of states iteratively. The set shrinks after each iteration and the process terminates when the size of the set cannot be reduced any more. The final set is a *fixpoint*, and includes all attractors in the initial set. If the initial set contains all states in the network, then all attractors in the network are discovered.

Algorithm 1 *Monolithic*(Z): Monolithic attractor detection on the set of Z of states

```
1: X ⇐ Z; Y ⇐ ∅;
2: while Y ≠ X do
3:     Y ⇐ X; X ⇐ Image(X, T);
4: end while
5: return X
```

Given a set Z of states in the network, Algorithm 1 computes the set X of successor states of Z under the transition relation T, and then successor states of X. This process is repeated until a fixpoint is reached. The fixpoint contains all attractors in Z. This algorithm utilises the property of synchronous BNs that every loop is an attractor because each state has only one outgoing transition. The following theorem shows the correctness of Algorithm 1.

Theorem 1. Given a set Z of states, let $\mathcal{A} = \{A_1, \ldots, A_m\}$ be the set of attractors in Z. Algorithm 1 returns the set X of states such that $X = A_1 \cup \ldots \cup A_m$.

Proof. First of all, we prove that each $A_i \in \mathcal{A}$ is contained in X. When the **while** loop starts, A_i is in X as $X = Z$. Let X' be the set of successor states returned by the $Image$ function. Each state s in A_i is contained in X' because there exists a state $s' \in A_i$ such that $s' \to s$. Therefore, we have $A_1 \cup \ldots \cup A_m \subseteq X'$.

Algorithm 2 *Search*(s): Search for the loop reachable from state s

1: $X \Leftarrow \{s\}$; $Y \Leftarrow \{s\}$;
2: **while** $Y \neq \emptyset$ **do**
3: $\quad Z \Leftarrow Image(Y, T)$; $Y \Leftarrow Z \setminus X$; $X \Leftarrow X \cup Y$;
4: **end while**
5: $A \Leftarrow Z$;
6: **while** $Z \neq \emptyset$ **do**
7: $\quad Z \Leftarrow Image(Z, T)$; $Z \Leftarrow Z \setminus A$; $A \Leftarrow A \cup Z$;
8: **end while**
9: **return** A

Now we prove that for each state $s \in X$, there exists a state $s' \in X$ such that $s' \to s$. This can be done by contradiction. Assume there is no such s' in X. Then we apply the function $Image$ again on X and obtain $X' = Image(X, T)$. Clearly, X' does not contain s, which is in conflict with the premise that X is a fixpoint.

Let $\overline{X} = X \setminus A_1 \cup \ldots \cup A_k$. Since the final set X is a fixpoint, from each state $s \in \overline{X}$, we can have an infinite sequence of states such that $\rho = s_0 s_1 s_2 \ldots$, where $s_0 = s$, $s_i \in X$, and $s_{i+1} \to s_i$ for all $i \geqslant 0$. Let k be the largest index in ρ such that $s_i \in \overline{X}$ for all $0 \leqslant i \leqslant k$ and $s_{k+1} \notin \overline{X}$. Assume $\overline{X} \neq \emptyset$. As X is finite, there must exist an attractor $A_j \in \mathcal{A}$ in ρ such that $s_{k+1} \in A_j$. Therefore, there are two outgoing transitions from s_{k+1}: one goes to s_k and the other to a state in A_j. This is in conflict with the semantics of synchronous BNs. Hence, $\overline{X} = \emptyset$.

4.2 Enumerative search attractor detection algorithm

Algorithm 1 begins with the whole set of states in the BN. Its performance can deteriorate dramatically as the size of the network increases. To remedy this, we propose an enumerative search strategy to deal with networks of large size. We first introduce an algorithm, i.e., Algorithm 2, for detecting a loop from an arbitrary initial state. Algorithm 2 computes the set X of states that can be reached from a state s and the attractor A contained in X. This algorithm iteratively uses forward reachability, i.e., the function $Image$, to compute A. In each iteration, the newly discovered successor states are stored in Z. Actually, Z contains only one state because each state in synchronous BNs has exactly one outgoing transition. The last newly discovered state is a state in the attractor, from which the whole attractor can be identified. This is performed in the second **while** loop of Algorithm 2. As an example of how this algorithm works, consider a part of the state space of some synchronous BN presented in Figure 2. After calling *Search(s1)*, Z in the first **while** loop will consecutively consist of s_2, s_3, s_4, s_5, s_6, and finally s_3 again. With this the first loop will end. The state s_3 is a member of the attractor $\{s_3, s_4, s_5, s_6\}$. This attractor is constructed iteratively in the second loop and returned as A.

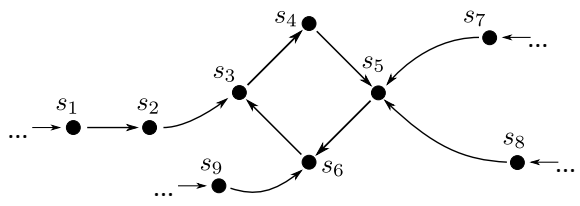

Figure 2 Part of the state space of some synchronous BN. The remaining states of the state space not shown in the figure are indicated by incoming arrows with ... to states s_1, s_7, s_8, and s_9. These arrows stand for potentially more than one incoming transition to the four states.

Algorithm 3 *Predecessor*(Z): Compute predecessors for the set Z of states

1: $Y \Leftarrow Z$; $X \Leftarrow Z$;
2: **while** $Y \neq \emptyset$ **do**
3: $\quad Y \Leftarrow Preimage(Y, T)$; $Y \Leftarrow Y \setminus X$;
4: $\quad X \Leftarrow X \cup Y$;
5: **end while**
6: **return** X

Algorithm 4 *SplitPre*(Z): Compute predecessors of the set Z of states in a divide-and-conquer manner

1: $X \Leftarrow Z$;
2: **for all** $s \in Z$ **do**
3: $\quad Y \Leftarrow Preimage(\{s\}, T)$;
4: $\quad Y \Leftarrow Y \setminus Z$; $X \Leftarrow X \cup Predecessor(Y)$;
5: **end for**
6: **return** X

After identifying the reachable attractor from an arbitrary initial state, we continue to compute the *predecessors* of the attractor. This is done with Algorithm 3, which can compute the set X of states that can reach a given set Z of states in the network via the transition relation. By slightly abusing the notation, the states in X are the predecessors of those in Z. X is computed by iteratively applying the *Preimage* function.

For any two states s and s' in a cyclic attractor A, their predecessors are the same ($Predecessor(s) = Predecessor(s') = Predecessor(A)$). Therefore, the above mentioned Algorithm 3 will repeat unnecessary *Preimage* operations when computing the predecessors for a cyclic attractor. We improve on this by considering individual states in Z, one at a time. For each individual state, the set of states that can reach it are computed as shown in Algorithm 4.

In Algorithm 4, the statement $Y \Leftarrow Preimage(\{s\}, T)$ computes the set Y of source states that can reach state s in one transition. As this algorithm is called in Algorithm 5 with the input Z being the set of states that form an attractor, Y contains not only transient states, but also a state s' in the attractor. Thus, we need the next statement $Y \Leftarrow Y \backslash Z$ to remove s' from Y because s' will be processed separately in the **for all** loop. As an example, we follow the steps of Algorithm 4 run on the attractor states in Figure 2, i.e., $SplitPre(\{s_3, s_4, s_5, s_6\})$. Each of the four attractor states is considered one by one. For s_3, we have $Preimage(s_3, T) = \{s_2, s_6\}$. The state s_6 is an attractor state, so it is not considered in Y which becomes $\{s_2\}$. Finally, the predecessors of Y are computed with the use of Algorithm 3, i.e., state s_1 and all its predecessors in the state space not shown in Figure 2. Similarly, for s_4 we have $Preimage(s_4, T) = \{s_3\}$, $Y = \emptyset$, and the predecessors of Y is an empty set. For s_5 we have $Preimage(s_5, T) = \{s_4, s_7, s_8\}$, $Y = \{s_7, s_8\}$, and the predecessors of Y are all the states in the state space that lead to states s_7 and s_8 (not shown in Figure 2). Finally, for s_6 we have $Preimage(s_6, T) = \{s_5, s_9\}$, $Y = \{s_9\}$, and the predecessors of Y are all the states in the state space that lead to state s_9 (not shown in Figure 2). The basic idea of Algorithm 4 is based on the following property of synchronous BNs: each state has only one outgoing transition. This guarantees that the sets returned by the *Predecessor* function in each iteration of the **for all** loop are pair-wise disjoint. In our above example, $Predecessor(\{s_2\})$, $Predecessor(\emptyset)$, $Predecessor(\{s_7, s_8\})$, and $Predecessor(\{s_9\})$ are all pair-wise disjoint.

With Algorithm 2 and Algorithm 4, we form our enumerative search strategy, shown in Algorithm 5, for detecting all attractors reachable from states in a set Z. Algorithm 5 starts with a randomly chosen state $s \in Z$ and computes the attractor A reachable from s. After removing from Z all backward-reachable states of A, the algorithm continues to search for other attractors of the remaining states in Z.

Algorithm 5 *Enumerative*(Z): Compute all attractors reachable from the set Z of states

1: $As \Leftarrow \emptyset$;
2: **while** $Z \neq \emptyset$ **do**
3: pick up a state $s \in Z$; $A \Leftarrow Search(s)$;
4: $Y \Leftarrow SplitPre(A)$; $As \Leftarrow As \cup A$; $Z \Leftarrow Z \setminus Y$;
5: **end while**
6: **return** As

The algorithm in GenYsis is very similar to Algorithm 5. The difference is that GenYsis computes all predecessors that reach an attractor in one fixpoint computation, i.e., using Algorithm 3. Algorithm 5 utilises Algorithm 4 instead. The intuition behind Algorithm 4 is that by splitting a simple attractor into individual states and by computing their predecessors separately it can better explore the regularity of the BDDs and accelerate the computation.

5 Accelerating via Decomposition

Both algorithms presented above search the entire state space of a BN, which could become very slow when the network is large and dense (see Section 7). In this section, we propose a decomposition attractor detection algorithm to reduce the size of the state space to be searched. Prior to detecting attractors of a BN, we first analyse the structure of the BN and try to decompose the BN into sub-networks, called *blocks*. The attractor detection is then performed on each block and the attractors detected are used to restrict the possible initial states of the original BN. The attractors of the original BN can then

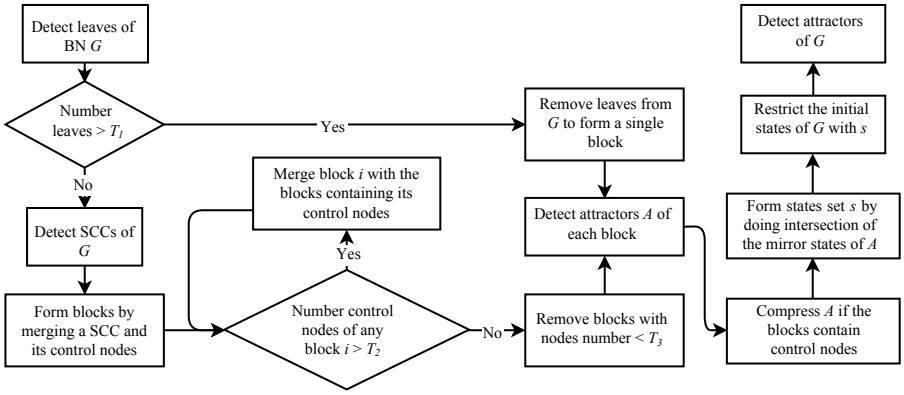

Figure 3 Flowchart of the decomposition attractor detection algorithm.

be detected by considering the restricted initial states only. The steps of this algorithm are shown in Figure 3. We now explain the algorithm in details using the following two cases.

Case 1: Leaf-based decomposition. We start by detecting a special type of nodes, called *leaf nodes* (or *leaves* for short) of a given BN G. Leaves are recursively defined as follows.

Definition 1. A node in a BN is a *leaf node* (or *leaf* for short) if and only if either (1) it has no child nodes or (2) it has no other children after iteratively removing all its child nodes which are leaf nodes.

An example of leaf nodes (leaves) is shown in Figure 4. Node b is affected by nodes a and c, and it does not have any children. Therefore, node b is a leaf. Node c contains only one child b and b is a leaf node. Hence, c is also a leaf node. But a is not a leaf, as a has itself as a child after removing b and c.

If the number of leaves in a BN exceeds the threshold value T_1, we simply remove all the leaves. The threshold value T_1 and the other two threshold values T_2 and T_3 which are used later in this section are set empirically. In our experiments, T_1 is set as $N/6$, where N is the total number of nodes in the BN. Both T_2 and T_3 are set to 2. After removing all the leaf nodes, the remaining nodes in the BN form a single block B. Formally, the blocks are defined as follows.

Definition 2. Given a BN $G(V, \boldsymbol{f})$ where $V = \{x_1, x_2, \ldots, x_n\}$ and $\boldsymbol{f} = \{f_1, f_2, \ldots, f_n\}$, a *block* $B(V^B, \boldsymbol{f^B})$ is a subset of the network, where $V^B \subseteq V$. For any node $x_i \in V^B$, if B contains all the parent nodes of x_i, its Boolean function in B remains the same as in G, i.e., f_i; otherwise, the Boolean function is undetermined, meaning that x_i is free to take either value 0 or 1 at any time point.

Definition 3. Given a state $\boldsymbol{x}^B$ of a block B, we can extend it to a state $\boldsymbol{x}$ of the original BN G by arbitrarily specifying the values of the nodes that are not in B, i.e., the nodes in $V \setminus V^B$. We call state $\boldsymbol{x}$ a *mirror* state of $\boldsymbol{x}^B$ and state $\boldsymbol{x}^B$ a *compressed* state of $\boldsymbol{x}$ in B. Let S_G be a set of states in G and S_B be a set of states in B. Denote the set of compressed states of S_G in block B as $\mathcal{C}_B(S_G)$ and the set of mirror states of S_B in G as $\mathcal{M}_G(S_B)$.

Notice that each state of B has more than one mirror state and each state of G has only one compressed state in B.

Definition 4. Given a BN G and a block B in G, let Φ be the set of attractor states of G and Φ^B be the set of attractor states of B. We say that B *preserves* the attractors of G if $\mathcal{C}_B(\Phi) \subseteq \Phi^B$.

Theorem 2. Given a BN G and a block B in G, let Φ be the set of attractor states of G and Φ^B be the set of attractor states of B. If B preserves the attractors of G, then $\Phi \subseteq \mathcal{M}_G(\Phi^B)$.

Proof. Since B preserves the attractors of G, $\mathcal{C}_B(\Phi) \subseteq \Phi^B$. By Definition 3, we have that $\Phi \subseteq \mathcal{M}_G(\mathcal{C}_B(\Phi))$. Hence, $\Phi \subseteq \mathcal{M}_G(\Phi^B)$.

Theorem 3. Given a BN G and a block B in G, let Φ^B be the set of attractor states of B. The set of reachable states from $\mathcal{M}_G(\Phi^B)$ in G is $\mathcal{M}_G(\Phi^B)$.

Proof. Let $\boldsymbol{x}$ be a state in $\mathcal{M}_G(\Phi^B)$. We have $\mathcal{C}_B(\{\boldsymbol{x}\}) \in \Phi^B$. Assume that state $\boldsymbol{y}$ is reachable from $\boldsymbol{x}$ by one transition of G. Since a node in B either has the same Boolean function as the corresponding node

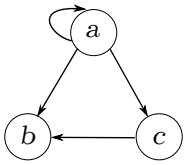

Figure 4 An example of leaves.

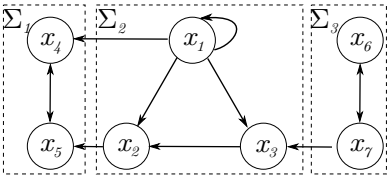

Figure 5 An example of SCC decomposition.

in G or is free to take any value at any time point, there is a transition of B from $\mathcal{C}_B(\{\boldsymbol{x}\})$ to $\mathcal{C}_B(\{\boldsymbol{y}\})$. Since $\mathcal{C}_B(\{\boldsymbol{x}\})$ is an attractor state of B, necessarily $\mathcal{C}_B(\{\boldsymbol{y}\}) \in \Phi^B$. Therefore, $\boldsymbol{y} \in \mathcal{M}_G(\mathcal{C}_B(\{\boldsymbol{y}\})) \subseteq \mathcal{M}_G(\Phi^B)$. By the arbitrary choice of $\boldsymbol{x}$ and by induction on the length of a path in the state transition system of G the claim of the theorem follows.

Since leaves have no affect on the nodes in the block, B preserves the attractors of G. According to Theorem 2, the mirror states of B's attractor states cover all G's attractor states. Therefore, by searching from the mirror states only instead of the whole state space, we can detect all the attractor states. In fact, by restricting the initial states to the mirror states, we restrict the state space to the mirror states as well since the reachable states equal to the mirror states according to Theorem 3.

Case 1 corresponds to the "Yes" branch of the first decision process in Figure 3. This branch is suitable for BNs which contain a large number of leaves or a large number of one-node SCCs which are not suitable for the "No" branch.

Case 2: SCC (strongly connected component)-based decomposition. If the number of leaves in a BN does not exceed T_1, we continue to decompose the network with SCC-based decomposition method. The decomposition starts by detecting SCCs of the BN. Figure 5 shows the decomposition of a BN into three SCCs: Σ_1, Σ_2, and Σ_3. A node outside an SCC that is a parent to a node in the SCC is referred to as a *control node* of this SCC. In Figure 5, nodes x_1 and x_2 are control nodes of Σ_1 and node x_7 is a control node of Σ_2. An SCC together with its control nodes forms a *block*. In such blocks, nodes belonging to an SCC are regulated by the same Boolean functions as in the original BN while control nodes are free to take any value at any time point.

Theorem 4. Given a BN G, let $\Sigma = \{\Sigma_1, \ldots, \Sigma_m\}$ be the set of SCCs in G and let $\mathcal{B} = \{B_1, \ldots, B_m\}$ be the set of blocks formed by merging an SCC with its control nodes. Block B_i preserves the attractors of G for any $1 \leqslant i \leqslant m$.

Proof. Let $\Phi = \{\phi_1, \ldots, \phi_n\}$ be the set of G's attractor states. We show that for any $\phi_j \in \Phi$ and for any $B_i \in \mathcal{B}$, the compressed state of ϕ_j of B_i is an attractor state of B_i. Let $\phi_j = \phi_{j_1}$ and let $A_j = (\phi_{j_1}, \phi_{j_2}, \ldots, \phi_{j_k})$ be the attractor that contains ϕ_j, where the states are ordered in accordance with the state transition sequence $\phi_{j_1} \to \phi_{j_2} \to \ldots \to \phi_{j_k} \to \phi_{j_1}$. In block B_i, the corresponding compressed states of A_j are $\mathcal{C}_{B_i}(A_j) = \big\{\mathcal{C}_{B_i}(\{\phi_{j_1}\}), \mathcal{C}_{B_i}(\{\phi_{j_2}\}), \ldots, \mathcal{C}_{B_i}(\{\phi_{j_k}\})\big\}$. For any state $\phi_{j_p} \in A_j$ and its compressed state $\mathcal{C}_{B_i}(\{\phi_{j_p}\})$, we let the control nodes in B_i to update their values in state $\mathcal{C}_{B_i}(\{\phi_{j_p}\})$ in the same way as in G in state ϕ_{j_p}. Since the nodes in the SCC of block B_i are regulated by the same Boolean functions as in the original BN, there is a transition of B_i from $\mathcal{C}_{B_i}(\{\phi_{j_p}\})$ to $\mathcal{C}_{B_i}(\{\phi_{j_{p+1}}\})$ (we let $\phi_{j_{p+1}} = \phi_{j_1}$ if $p = k$). Therefore, $\mathcal{C}_{B_i}(A_j)$ forms an attractor of B_i with the transition sequence $\mathcal{C}_{B_i}(\{\phi_{j_1}\}) \to \mathcal{C}_{B_i}(\{\phi_{j_2}\}) \to \ldots \to \mathcal{C}_{B_i}(\{\phi_{j_k}\}) \to \mathcal{C}_{B_i}(\{\phi_{j_1}\})$. Hence, B_i preserves the attractors of G.

Definition 5. Given a BN G, B is a block of G formed by merging an SCC with its control nodes. We further compress the attractor states of B by removing the values of the control nodes and call these states as *double compressed* attractor states.

The set of attractor states of a block is a subset of the mirror states of the double compressed attractor states of the block.

Theorem 5. Given a BN G, let $\Sigma = \{\Sigma_1, \ldots, \Sigma_m\}$ be the set of SCCs in G and let $\mathcal{B} = \{B_1, \ldots, B_m\}$ be a set of blocks, where B_i $(i = 1, \ldots, n)$ is a block formed by merging the SCC Σ_i with its control nodes. Let Φ be the set of attractor states of G and let $\mathcal{A} = \{A_1, \ldots, A_m\}$ be the family of sets of double

compressed attractor states of each block. It holds that $\Phi \subseteq \cap_{i=1}^{m} \mathcal{M}_G(A_i)$.

Proof. Denote Φ^{B_i} the set of attractor states of block B_i for $1 \leqslant i \leqslant m$. Since $\Phi^{B_i} \subseteq \mathcal{M}_{B_i}(A_i)$, we have $\mathcal{M}_G\{\Phi^{B_i}\} \subseteq \mathcal{M}_G(\mathcal{M}_{B_i}(A_i))$ for $1 \leqslant i \leqslant m$. By Theorems 4 and 2, $\Phi \subseteq \mathcal{M}_G(\Phi^{B_i})$ for all $1 \leqslant i \leqslant m$. Therefore, $\Phi \subseteq \cap_{i=1}^{m} \mathcal{M}_G(\Phi^{B_i}) \subseteq \cap_{i=1}^{m} \mathcal{M}_G(\mathcal{M}_{B_i}(A_i))$. According to Definition 3, we have that $\mathcal{M}_G(A_i) = \mathcal{M}_G(\mathcal{M}_{B_i}(A_i))$ for all $1 \leqslant i \leqslant m$. Hence, it holds that $\Phi \subseteq \cap_{i=1}^{m} \mathcal{M}_G(A_i)$.

After computing the double compressed attractors of each block, we get the sets of mirror states of the original BN. By Theorem 5, we can use the intersection of these sets to reduce the search space.

Case 2 corresponds to the "No" branch of the first decision process. This branch contains a decision process for merging two blocks and a process for removing blocks with node number smaller than T_3. The decision process is to guarantee that the number of control nodes in a block is as small as possible. Adding q control nodes to an SCC does not only increases the state space but also adds to each state 2^q transitions, most of which do not exist in the original BN. This exponentially increases the time cost for computing forward and backward states. Therefore, reducing the number of control nodes can potentially reduce the time cost for computing the attractors of a block. Note that Theorem 4 and Theorem 5 still hold for the merged blocks since the merged blocks contain the same types of nodes as the blocks used in the two theorems, i.e., nodes regulated by the same Boolean functions as in the original BN and control nodes that are free to take any value at any time point. The process of removing blocks eliminates small blocks as attractors of small blocks will much likely lead to mirror states being virtually the whole state space of the original BN. Note that Theorem 5 still holds after removing small blocks since the intersection of sets of mirror states is used in the theorem.

The detection of attractors for a Boolean network or a block in Figure 3 is performed using one of the two algorithms mentioned in Section 4. In general, for the leaf-based decomposition branch, the enumerative search algorithm works faster and for the SCC-based decomposition branch, the monolithic search algorithm is preferable.

6 Heuristics on BDD Variable Orders

The order of the input variables of a BDD can largely affect the constructed BDD size and the corresponding operations [3]. In consequence, the performance of the BDD attractor detection algorithm also heavily relies on the BDD variable orders. It is known to be an NP-complete problem [2] to find an optimal variable ordering for constructing a minimum-size BDD. However, a lot of efficient heuristics have been proposed in literature to establish near-optimal variable orderings for BDDs. In general, there are two types of heuristic techniques for establishing efficient variable orders for BDD: static variable ordering and dynamic variable ordering. Static variable ordering forms an order before the construction of the BDD while dynamic variable ordering attempts to form an optimal order dynamically during the construction process. As the order is formed before construction of the BDD, static variable ordering cannot guarantee a good order. On the contrary, dynamic variable ordering allows adjusting the variable order during the construction of the BDD, which is more effective in finding a good variable order. However, this dynamic reordering process is usually time-consuming and therefore less practical. In this paper, we concentrate on the static variable ordering heuristics and propose two BDD order heuristics based on the structure of BNs.

Each node of a BN is represented as one variable in the corresponding BDD. For simplicity, we treat ordering the variables of a BDD as ordering the nodes of a BN. As shown in [4], variables that are topologically close in the original network should be ordered relatively close to each other in the BDD. In a BN, a node is closer to its parent nodes and its child nodes comparing to others. Considering this, we form the following two heuristics: (I) put a node's parent nodes right after the node; (II) put a node's child nodes right after the node.

For heuristic (I), we order the nodes as follows. (1) From all the unordered nodes, find the one with the fewest child nodes and order it as the next one. (2) For each of the ordered nodes, find its unordered parent nodes and order them as the next ones. If a node has more than one unordered parent node, the

order of its parent nodes can be arbitrary. (3) Repeat Step 2 until all the parent nodes of ordered nodes have been ordered. (4) Finish if all the nodes have been ordered; otherwise, go back to step (1).

For heuristic (II), we order the nodes as follows. (1) From all the unordered nodes, find the one with the most child nodes and order it as the next one. (2) For each of the ordered nodes, find its unordered child nodes and order them as the next ones. If a node has more than one unordered child node, the order of its child nodes can be arbitrary. (3) Repeat Step (2) until all the child nodes of ordered nodes have been ordered. (4) Finish if all the nodes have been ordered; otherwise, go back to step (1).

Heuristic (I) will put the node which has the fewest child nodes in the beginning; while, on the opposite, heuristic (II) will put the node which has the most child nodes in the beginning. The node with the most children nodes is considered as the most influential node in the BN. It is suggested in [17] that the more influential node should be put earlier in the ordering. However, in the case of BNs, our experience leads us to the conclusion that topological closeness plays a more important role than the influence of nodes. This can be demonstrated by the experimental results in Section 7.

7 Evaluation

We have implemented the algorithms presented in Section 4 in the model checker MCMAS [15]. In this section, we demonstrate with experiments that our algorithms can outperform the existing BDD-based attractor detection algorithm in [8] and our proposed BDD variable order heuristics can improve the performance even further. Comparison with other SAT-based methods is one of our future works.

All the experiments in this paper are conducted on a high-performance computing (HPC) cluster with Intel Xeon L5640@2.26GHz processor. The BNs we use for the experiments are randomly generated with ASSA-PBN [18], which can generate random BNs complying with specified parameters, e.g., the number of nodes in the BN and the maximal number of variables of the predictor functions.

7.1 Evaluation for small BNs

We first compare MCMAS with GenYsis for detecting attractors of small BNs. We generate BNs with the number of nodes from the set $\{20, 30, 40, 50\}$ and for each number in the set, we generate three BNs of different densities. We divide the BNs into three different classes with respect to their densities: class d, short for dense, refers to BNs whose density is over 3.1; class i, short for in-between, refers to BNs whose density is in the range [2.0, 3.1]; and class s, short for sparse, refers to BNs whose density is in the range [1.0, 2.0). Note that if the density of a BN is below 1.0, there exists at least one node without any predictor function (see Section 3.1). We do not consider BNs containing such nodes since they can be removed from the BN without changing the behaviour of the BN.

We name models in accordance with the following format "number of nodes_density class_model index". We show in Table 1 the number of detected singleton and cyclic attractors and the time cost of the two tools (in seconds). To compare the performance of the proposed BDD orders, we also run the attractor detection algorithms with different BDD orderings. We set the maximum running time to one hour and mark as "—" in the table if the program fails to detect the attractors within this time bound.

It is clear from Table 1 that using the proposed monolithic attractor detection algorithm MCMAS performs much faster than GenYsis in detecting attractors for small BNs. Our proposed BDD variable order heuristics can further improve the performance in most cases. In particular, MCMAS reduces the time cost by about 81 times for the 30-node dense BN by ordering the BDD variables with heuristic (II); GenYsis fails to handle dense BNs of more than 40 nodes while MCMAS still manages to handle those BNs. In general, the monolithic algorithm gains better performance for BNs with node number less than 50. For large models, we shall use different algorithms and we demonstrate the results in the next section. In fact, BNs with around 50 nodes seem to be the borderline models. In our evaluation, for BNs with 50 nodes, the monolithic algorithm (Algorithm 1) performs better for the dense and in-between networks, while the enumerative algorithm (Algorithm 5) performs better for the sparse ones (see the performance of MCMAS for 50-node BNs in Table 1 and Table 2).

Table 1 Performance of MCMAS and GenYsis for small BNs. The time is in seconds. Gen. is short for GenYsis; sin. is short for singleton; cyc. is short for cyclic; ran. is short for random; and heu. is short for heuristic.

model name	# attractor		MCMAS monolithic detection time			Gen. time	model name	# attractor		MCMAS monolithic detection time			Gen. time
	sin.	cyc.	ran. order	heu. (I)	heu. (II)			sin.	cyc.	ran. order	heu. (I)	heu. (II)	
20_d_01	0	1	0.13	0.09	0.12	0.65	40_d_07	0	2	407.38	302.89	576.26	—
20_i_02	0	4	0.04	0.04	0.06	0.32	40_i_08	4	4	23.90	24.39	25.70	64.77
20_s_03	0	3	0.04	0.03	0.03	0.08	40_s_09	2	12	2.51	1.33	3.30	6.81
30_d_04	1	0	5.25	6.81	3.80	243.56	50_d_10	0	1	270.87	360.75	391.32	—
30_i_05	0	10	0.59	0.45	0.48	4.62	50_i_11	1	0	181.83	398.96	121.86	—
30_s_06	0	3	0.22	0.22	0.22	1.02	50_s_12	1	29	1.179	1.261	1.048	7.91

7.2 Evaluation for large BNs

For large BNs, MCMAS applies the enumerative algorithm (Algorithm 5) and the decomposition attractor detection algorithm (Figure 3). We demonstrate the efficiency of these two algorithms by experiments on a number of large BNs.

We present in Table 2 the time cost of MCMAS and GenYsis for detecting attractors of large BNs with node number in the set $\{50, 60, 80, 100, 120, 300, 400\}$. For each of the node numbers, three BNs with different densities are generated. With the node number increasing, it becomes difficult for both the MCMAS and GenYsis to detect attractors within one hour. Therefore, we increase the time to three hours. We omit the data if both methods fail to detect the attractors within this limit. In total, we obtain data for 12 (out of 21) BNs.

In 11 out of these 12 cases, MCMAS is faster than GenYsis using the enumerative search strategy and a random BDD order. Moreover, if we order the BDD variables according to heuristic (I), MCMAS performs better than GenYsis in all the 12 cases. Notably, GenYsis fails to handle the 80-node sparse BN while MCMAS still detects its attractors in 6.09 seconds using heuristic (I). As the performance of BDD operations is sensitive to BDD orders, it would be better if we could compare the performance of MCMAS and GenYsis under the condition that they use the same BDD variables order. However, the source code of GenYsis is not available and the BDD variables order used in GenYsis is unclear to us. Nevertheless, it is still clear from Table 2 that MCMAS performs better than GenYsis for large BNs using the enumerative algorithm and the BDD variable order heuristic (I).

In 11 out of the 12 compared cases, heuristic (I) results in a better performance of MCMAS than the random order; while for heuristic (II), the number of cases is 7 out of 12. Besides, heuristic (I) performs better than heuristic (II) in 8 out of 12 cases. In general, both heuristics can improve the performance of MCMAS for large BNs, but heuristic (I) often performs better.

The decomposition detection algorithm works faster than the enumerative search detection algorithm in most cases. However, this does not mean that the enumerative search detection algorithm is meaningless. In fact, the decomposition detection algorithm is based on the enumerative search detection algorithm. The influence of order on the performance is not as impressive as that in the case of the enumerative search algorithm. Heuristic (II) shows better performance in 7 out of 12 cases. In these 12 cases, the last three cases are performed by the leaf-based decomposition while the others are performed by SCC-based decomposition.

The missing data in Table 2 are mainly for dense and in-between networks. Both MCMAS and GenYsis fail to handle dense BNs with over 80 nodes and in-between networks with over 100 nodes in 3 hours. In general, increasing the density exponentially increases the size of Boolean functions in the BNs and the corresponding BDD representation becomes more complex. This is the reason why the time cost for dense networks is much larger than that of the sparse networks in both Table 1 and Table 2. Developing efficient algorithms to handle large dense models is part of our future work.

Table 2 Performance of MCMAS and GenYsis for large BNs.

model name	# attractor		MCMAS enumerative search detection time(s)			MCMAS decomposition detection time(s)			GenYsis time(s)
	sin.	cyc.	random order	heuristic (I)	heuristic (II)	random order	heuristic (I)	heuristic (II)	
50_*d*_10	0	1	3070.11	1341.28	5398.86	304.63	549.43	598.11	—
50_*i*_11	1	0	6683.67	915.516	2801.74	284.34	319.40	300.55	—
50_*s*_12	1	29	1.02	0.87	1.16	0.85	1.01	0.67	7.91
60_*d*_13	0	5	2032.16	3117.18	1236.45	795.01	1254.88	591.17	—
60_*i*_14	1	1	560.95	67.86	542.39	27.35	25.79	28.74	120.23
60_*s*_15	0	5	0.51	0.22	0.27	0.4	0.27	0.29	0.99
80_*i*_16	6	124	224.94	84.22	248.54	70.12	33.88	46.02	275.84
80_*s*_17	0	4	9.99	6.09	11.82	1.64	1.49	1.49	—
100_*s*_18	1	13	12.65	12.15	10.61	10.13	4.47	4.67	54.02
120_*s*_19	0	16	24.72	14.76	33.15	6.39	6.71	6.72	38.79
300_*s*_20	0	64	96.57	65.07	81.42	70.34	86.27	77.75	5895.26
400_*s*_21	3	350	2771.50	2686.77	2467.11	2123.42	2131.13	2106.52	5315.84

8 Biological Case Study: A Signalling Network of Apoptosis

Cells in a multicellular organism are tightly controlled by mechanisms that regulate the cell division and the cell death. One of these mechanisms is *apoptosis*, the programmed cell death: if cells are damaged or no longer needed, the intracellular death program is activated, which leads to cell death without lysis or damage to neighbouring cells. This process is regulated by a number of signalling pathways which are extensively linked by cross-talk interactions and which form an intricate system. In [22], a large-scale multi-value logic model of apoptosis in hepatocytes was presented, where all interactions were introduced based on literature study. The model was recast into the probabilistic Boolean network (PBN) framework, fitted to experimental steady-state data and analysed in [25]. Further, more quantitative analysis was performed with the ASSA-PBN tool, which enabled the computation of certain long-run influences and sensitivities of the PBN model, see [19] for more details.

In this work, the original model was recast into the Boolean network framework: a binary BN model which comprises 97 nodes (state-space of size 2^{97}). The BN model contains 10 input nodes. In the original model three levels of activation of FasL and UV nodes are considered represented by values 0, 1, and 2. We model this with two binary nodes, i.e., (FasL(1), FasL(2)) and (UV(1), UV(2)), respectively. The possible combinations of activations are: $(0,0)$ (level 0), $(1,0)$ (level 1), and $(0,1)$ (level 2). The combination $(1,1)$ is biologically meaningless and excluded from discussion. There is also a housekeeping node which value is fixed to 1 and which is used to model constitutive activation of certain nodes in the network. For the wiring of the BN model, see Figure 6 in the appendix.

In [22], the steady-state analysis was performed after excluding 13 interactions in order to break all the feedback loops in the network. Existence of feedback loops is a necessary condition for oscillations to be present in a system. By removing them, it was made sure that oscillations would not lead to ambiguous interpretation of the logical steady states. Here, we put back all the 13 interactions (cf. Table S4 in [22]) and consider the full model of apoptosis. We then apply our algorithms to compute all the attractors of the BN model: 2272 attractors are detected for all possible combinations of stimulation of the 10 input nodes and the housekeeping node. It takes 1559s, 681s, and 153s, respectively, to finish the computation with our monolithic, enumerative and decomposition algorithms. Note that GenYsis uses 3314 seconds to finish the same computation. In the following, we consider only biologically meaningful stimulations. In particular, we repeat the experiment of [22] and we focus on all single and double input-node stimulation scenarios and demonstrate the results for them in Table 3. We compare them with the results presented in Table 4 in [22]. For each scenario the corresponding one or two input node(s) are set to 1, the

Table 3 Apoptosis node value for all single and double input-node stimulation scenarios of the full Boolean model of apoptosis originally published in [22] with all feedback loops included. The smac-mimetics input node is denoted as s-m. The values 0 and 1 represent the steady-state value of the apoptosis node which is fixed to the respective value in the attractor(s). Oscillations of the apoptosis node value are denoted $\sim$, which represents an attractor of length four in which the apoptosis node takes values $1 \to 0 \to 0 \to 0$. The number in the parenthesis is the number of attractors for a particular stimulation, i.e., the number of attractors that can be reached dependent on the initial state of the non-input nodes in the network. Abberations with respect to the values presented in Table 4 in [22] are highlighted bold.

	Glucagon	Insulin	TNF	FasL(1)	FasL(2)	T2RL	IL-1	s-m	UV(1)	UV(2)
Glucagon	0 (7)	0 (7)	0 (7)	0 (1)	$\sim$ (1)	1 (1)	0 (7)	**1** (1)	1 (1)	**1** (1)
Insulin		0 (7)	0 (7)	0 (1)	$\sim$ (1)	1 (1)	0 (7)	**1** (1)	**1** (1)	**1** (1)
TNF			0 (7)	0 (1)	$\sim$ (1)	1 (1)	0 (7)	1 (1)	1 (1)	1 (1)
FasL(1)				0 (1)	—	1 (1)	0 (1)	1 (1)	1 (1)	1 (1)
FasL(2)					$\sim$ (1)	$\sim$ (1)	$\sim$ (1)	$\sim$ (1)	$\sim$ (1)	$\sim$ (1)
T2RL						1 (1)	1 (1)	1 (1)	1 (1)	1 (1)
IL-1							0 (7)	**1** (1)	**1** (1)	**1** (1)
s-m								**1** (1)	1 (1)	**1** (1)
UV(1)									1 (1)	—
UV(2)										**1** (1)

housekeeping node is fixed to 1, all the other input nodes are kept 0, and the attractors for all possible initial settings of the remaining nodes in the network are considered. There is either a single attractor or there are seven different attractors for each stimulation scenario. All attractors for all scenarios are of length four. As can be seen in Table 3, in all scenarios not involving activation of the FasL(2) input node, the apoptosis node in all the attractors is fixed either to 0 or to 1, which is inline with the fact that commitment to apoptosis is a digital (YES or NO) response of the network. Although some nodes in the network oscillate, the apoptosis node remains fixed. However, in all the scenarios where FasL(2) is active, the value of the apoptosis node oscillates and it values evolve cyclically in accordance with the pattern $1 \to 0 \to 0 \to 0$. The apoptosis node is inactive in three out of the four attractor states. Whether the apoptosis node oscillates or not depends on the dynamics of C3ap17 and C3ap17_2. From the wiring diagram of the BN model it can be observed that the constant activation of FasL(2) causes the value of the node C8a_2 to be always 1. In consequence, C3ap20 is fixed to 0 independent of the value of C8a. Moreover, C3ap20_2 is constantly 1. Hence, the values of C3ap17 and C3ap17_2 become independent of the values of C3ap20 and C3ap20_2. This breaks the feedback loop via C6. It can be observed that C3ap17_2 is negatively regulated by NFκB via XIAP_2 and both C3ap17 and C3ap17_2 (via an OR gate) negatively regulate C3ap17 through C3a_XIAP and BIR 1-2. Furthermore, there is no feedback from C3ap17 and C3ap17_2 that would influence the behaviour of NFκB. If NFκB were fixed to 0, C3ap17 would be fixed to 0 and C3ap17_2 to 1. If NFκB were fixed to 1, C3ap17_2 would be fixed to 0 and although C3ap17 would oscillate, the apoptosis node would be fixed to 0 through gelsolin. In this way the oscillations of the apoptosis node are induced by the oscillations of NFκB.

If FasL(2) is excluded, we observe that in comparison to the model without feedbacks, the apoptosis node is constantly activated whenever any of smac-mimetics, UV(1), or UV(2) input nodes are active. The differences are indicated by bold in the respective columns of Table 3. The biological meaning of these differences and the oscillations in the case of FasL(2) activation require further investigation.

9 Conclusions and Future Work

In this paper, we have resented several strategies for improving the BDD-based attractor detection algorithm shown in [8] for synchronous BNs, including (1) a monolithic algorithm for small BNs; (2) an enumerative algorithm for large BNs; (3) a decomposition method to accelerate attractor detection; and (4) two heuristics on ordering BDD variables. Experimental results show that the proposed strategies can improve the BDD-based attractor detection algorithm [8] for both small and large BNs. Comparing our algorithms with SAT-based approaches (e.g., see [5]) is one of our future works.

Acknowledgements Honyang Qu is supported by the EPSRC project EP/J011894/2 and the Royal Society project IE141180. Qixia Yuan is supported by the National Research Fund, Luxembourg (grant 7814267).

Conflict of interest The authors declare that they have no conflict of interest.

References

1 Akers, S B. Binary decision diagrams. IEEE Trans Comput, 1978, 100(6): 509–516
2 Bollig B, Wegener L. Improving the variable ordering of OBDDs is NP-complete. IEEE Trans Comput, 1996, 45(9): 993–1002
3 Bryant R E. Symbolic Boolean manipulation with ordered binary-decision diagrams. ACM Comput Surv, 1992, 24(3): 293–318
4 Drechsler R. Verification of multi-valued logic networks. In: Proc. 26th Symposium on Multiple-Valued Logic. IEEE, 1996. 10–15
5 Dubrova E, Teslenko M. A SAT-based algorithm for finding attractors in synchronous Boolean networks. IEEE/ACM Trans Comput Biol Bioinf, 2011, 8(5): 1393–1399
6 Dubrova E, Teslenko M, Martinelli A. Kauffman networks: Analysis and applications. In: Proc. 2005 IEEE/ACM International Conference on Computer-Aided Design. IEEE CS, 2005. 479–484
7 Garg A, Di Cara A, and Xenarios L, et al. Synchronous versus asynchronous modeling of gene regulatory networks. Bioinformatics, 2008, 24(17): 1917–1925
8 Garg A, Xenarios L, Mendoza L, et al. An efficient method for dynamic analysis of gene regulatory networks and in silico gene perturbation experiments. In: Proc. 11th Annual Conference on Research in Computational Molecular Biology. volume 4453 of LNCS. Springer, 2007. 62–76
9 Guo W, Yang G, Wu W, et al. A parallel attractor finding algorithm based on Boolean satisfiability for genetic regulatory networks. PLOS ONE, 2014, 9(4): e94258
10 Huang S. Genomics, complexity and drug discovery: insights from Boolean network models of cellular regulation. Pharmacogenomics, 2001, 2(3): 203–222
11 Irons D J. Improving the efficiency of attractor cycle identification in Boolean networks. Phys D, 2006, 217(1): 7–21
12 Kauffman S. Homeostasis and differentiation in random genetic control networks. Nature, 1969, 224: 177–178
13 Kauffman S. Metabolic stability and epigenesis in randomly constructed genetic nets. J Theor Biol, 1969, 22(3): 437–467
14 Lee C. Representation of switching circuits by binary-decision programs. Bell Syst Tech J, 1959, 38(4): 985–999
15 Lomuscio A, Qu H, Raimondi F. MCMAS: An open-source model checker for the verification of multi-agent systems. Int J Softw Tools Technol Transf, 2015, 1–22
16 Raeymaekers L. Dynamics of Boolean networks controlled by biologically meaningful functions. J Theor Biol, 2002, 218(3): 331–341
17 Malik S, Wang A R, Brayton R K, et al. Logic verification using binary decision diagrams in a logic synthesis environment. In: Proc. IEEE International Conference on Computer-Aided Design. IEEE, 1988. 6–9
18 Mizera A, Pang J, Yuan Q. ASSA-PBN: a tool for approximate steady-state analysis of large probabilistic Boolean networks. In: Proc. 13th International Symposium on Automated Technology for Verification and Analysis. volume 9346 of LNCS. Springer, 2015. 214–220. Software available at http://satoss.uni.lu/software/ASSA-PBN/
19 Mizera A, Pang J, Yuan Q. Reviving the two-state Markov chain approach (technical report). 2015. Available online at http://arxiv.org/abs/1501.01779
20 Mushthofa M, Torres G, Van de Peer Y, et al. ASP-G: an ASP-based method for finding attractors in genetic regulatory networks. Bioinformatics, 2014, 30(21): 3086–3092
21 Needham C J, Manfield I W, Bulpitt A J, et al. From gene expression to gene regulatory networks in Arabidopsis thaliana. BMC Syst Biol, 2009, 3(1): 85
22 Schlatter R, Schmich K, Vizcarra I A, et al. ON/OFF and beyond - a Boolean model of apoptosis. PLoS Comput Biol, 2009, 5(12): e1000595
23 Shmulevich I, Edward R D. Probabilistic Boolean networks: the modeling and control of gene regulatory networks. SIAM Press, 2010
24 Somogyi R, Greller L D. The dynamics of molecular networks: applications to therapeutic discovery. Drug Discov Today, 2001, 6(24): 1267–1277
25 Trairatphisan P, Mizera A, Pang J, et al. optPBN: An optimisation toolbox for probabilistic Boolean networks. PLOS ONE, 2014, 9(7): e98001
26 Zhao Y, Kim J, Filippone M. Aggregation algorithm towards large-scale Boolean network analysis. IEEE Trans Automat Contr, 2013, 58(8: 1976–1985
27 Zheng D, Yang G, Li X, et al. An efficient algorithm for computing attractors of synchronous and asynchronous Boolean networks. PLOS ONE, 2013, 8(4): e60593

Appendix: The Boolean Model of Apoptosis

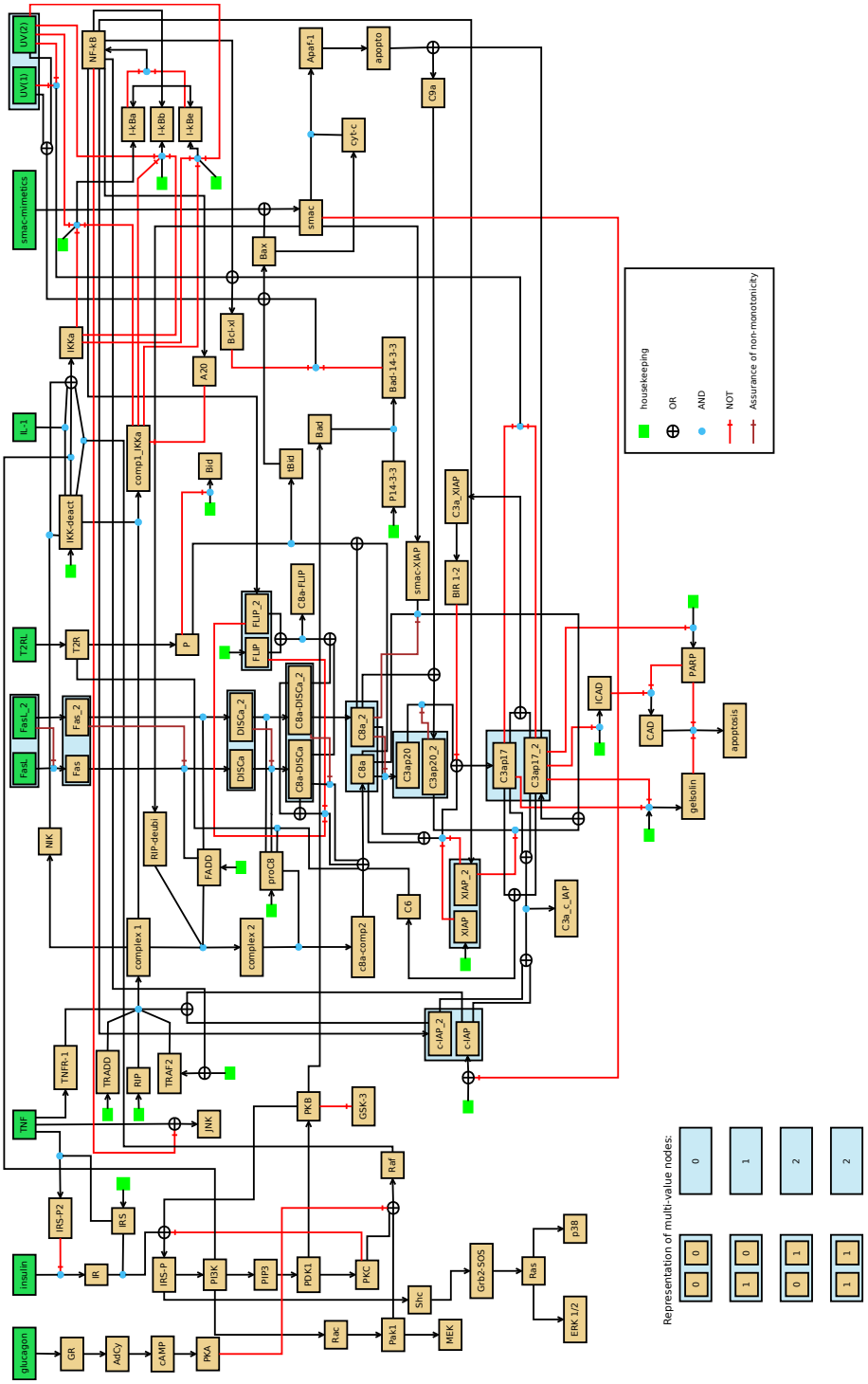

Figure 6 The wiring of the multi-value logic model of apoptosis by Schlatter et al. [22] recast into a binary Boolean network. For clarity of the diagram the nodes I-kBa, I-kBb, and I-kBe have two positive inputs. The inputs are interpreted as connected via $\oplus$ (logical OR).