Improving BDD-based Attractor Detection for Synchronous Boolean Networks

Hongyang Qu*
Department of Automatic Control & Systems Engineering, University of Sheffield, UK
h.qu@sheffield.ac.uk

Qixia Yuan†
Faculty of Science, Technology and Communication, University of Luxembourg, Luxembourg
qixia.yuan@uni.lu

Jun Pang‡
Faculty of Science, Technology and Communication, University of Luxembourg, Luxembourg
jun.pang@uni.lu

Andrzej Mizera
Faculty of Science, Technology and Communication, University of Luxembourg, Luxembourg
andrzej.mizera@uni.lu

ABSTRACT

Boolean networks are an important formalism for modelling biological systems and have attracted much attention in recent years. An important direction in Boolean networks is to exhaustively find attractors, which represent steady states when a biological network evolves for a long term. 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, and two heuristics on ordering BDD variables. We demonstrate the performance of our approach on a number of examples, and compare it with one existing technique in the literature.

Categories and Subject Descriptors

G.1.0 [**Mathematics of Computing**]: General—*Numerical algorithms*; D.2.4 [**Software**]: Software/Program Verification —*Formal methods*

General Terms

Algorithms

Keywords

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

*Supported by the EPSRC project EP/J011894/2 and the Royal Society project IE141180.
†Supported by the National Research Fund, Luxembourg (grant 7814267).
‡Partially supported by the State Key Laboratory for Novel Software Technology at Nanjing University, China.

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. The expression of genes is a fundamental process in living cells, both eukaryotic and prokaryotic. It has attracted much attention to acquire insights into the dynamics of GRNs.

Many formalisms have been proposed in the literature to study the dynamics of GRNs, including directed graphs, Bayesian networks, Boolean networks (BNs) [13] and probabilistic Boolean networks (PBNs) [23, 25], ordinary and partial differential equations, stochastic equations, and rule-based formalisms. Among them, BNs are broadly applied to represent GRNs, e.g., see [21, 7]. BN models the state of a gene as binary values and the interaction between genes as Boolean functions, known as predictor functions. BN reduces the complexity of a network by avoiding the expensive model of the kinetic information in the network and grasp only the wiring of the network. Despite of the highly simplified representation of the model, BN is still able to cope with one of the most important dynamic properties in GRN, i.e., *attractors*. Attractors are hypothesised to characterise cellular phenotypes [13] and correspond to functional cellular states such as proliferation, apoptosis, or differentiation [11].

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 demonstrated in [10] for synchronous BNs, which was developed by Abhishek et al. in 2007. The algorithm starts from one (randomly selected) initial state and detects attractors by computing the successor states and predecessor states of this initial state. It works well for small BNs, but becomes inefficient for large BNs. We propose to use different attractor detection strategies for networks of different sizes to improve the efficiency of the BDD-based approach. For small BNs, we propose to use a monolithic algorithm which searches for attractors from the whole state space, instead of

one initial state, to reduce the time for attractor detection. For large BNs, we improve the algorithm in [10] 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 detecting other attractors. Moreover, as the performance of BDD operations is largely affected by the order of BDD variables, we propose two heuristics on ordering BDD variables based on the structure of BNs to further improve the attractor detection algorithm. We have implemented our algorithms and the BDD variable order heuristics in the tool MCMAS [16] and demonstrated with extensive experiments that our proposed algorithms and heuristics can improve the efficiency of the BDD-based attractor detection algorithm and outperform the existing tool GenYsis [10].

2. RELATED WORK

In this section we review algorithms 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 therefore 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 [17]. However, this technique cannot guarantee to find all the attractors of a BN. Irons in [12] proposed a method by analysing partial states involving parts of the nodes. This method improves the efficiency 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 still 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, 10, 9] 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. in [10] proposed a method for detecting attractors for asynchronous BNs. Later, in [9], the synchronous BNs were considered and a combined synchronous-asynchronous modelling approach was proposed to improve the performance of attractor detection algorithms in asynchronous BNs. In a recent work [26], Zheng et al. developed an algorithms 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 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 and the number of nodes in the GRN.

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 simple model class for the analysis of the dynamical properties of gene regulatory networks [14], where projection of gene states to a 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 between 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.,

$$x_i(t+1) = f_i(x_{i_1}(t), x_{i_2}(t), \ldots, x_{i_{k(i)}}(t)), \tag{1}$$

where $1 \leq i_1 < i_2 < \cdots < i_{k(i)} \leq n$. 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 $1 \leq j \leq k(i)$ 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., activator set which contains genes that can activate x_i and 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., [9, 20]). Here the following three types of Boolean functions are considered:

$$x_i(t+1) = \left(\bigvee_{j=1}^{m} x_j^a(t)\right) \wedge \neg \left(\bigvee_{j=1}^{n} x_j^{in}(t)\right), \tag{2}$$

$$x_i(t+1) = \left(\bigvee_{j=1}^{m} x_j^a(t)\right), \tag{3}$$

$$x_i(t+1) = \neg \left(\bigvee_{j=1}^{n} x_j^{in}(t)\right), \tag{4}$$

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. Equation 2 is used when there are both activators and inhibitors for a gene x_i. Equation 3 is used when gene x_i only has activators as its parent nodes

and Equation 4 is used in the case that 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 updates 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 states of a synchronous BN, i.e.,

$$\boldsymbol{x}(t+1) = \boldsymbol{f}(\boldsymbol{x}(t)). \tag{5}$$

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 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 [13].

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 [15] 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 evaluations that give the Boolean function true; while a path from the root node to the 0-terminal represents variable evaluations that give the Boolean function false. For example, a BDD represents the Boolean function $f = (a \text{ OR } c) \text{ AND } b$ is shown in Figure 1a. In this figure, root node and terminal nodes are represented with squares while decision nodes are represented with circles. A solid arrow starting from an decision node represents an evaluation of 1 for that node while a dashed arrow starting from an decision node represents an evaluation of 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 those unnecessary variables for the above BDD, we get a reduced BDD, as shown in Figure 1b. 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., $a = 1,\ b = 1,\ c = 1$ and $a = 1,\ b = 1,\ c = 0$; 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 1c, we could 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]. As a consequence, using heuristic 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

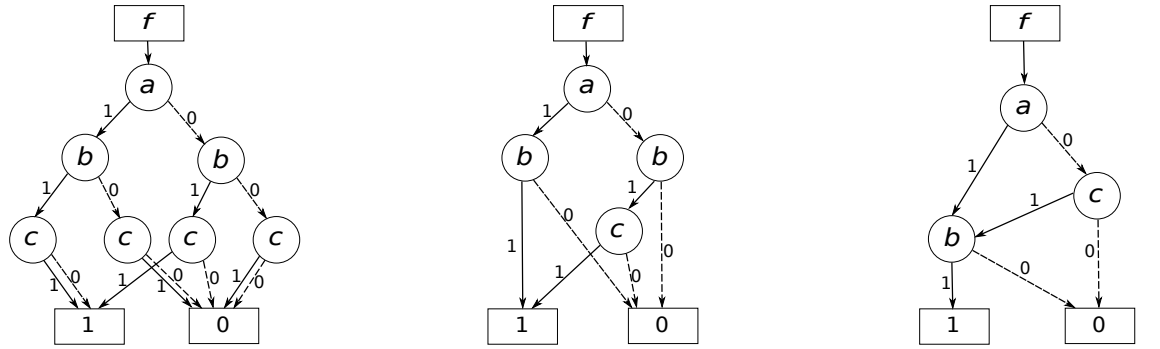
Boolean networks can be easily modelled as *transition systems*, which then can be encoded in BDDs. A transition system TS is a tuple $\langle S, S_0, T\rangle$ where

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 among states. In a transition $(s_1, s_2) \in T$, state s_1 is called *source state* and s_2 *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. Therefore, in the transition system, we have $S_0 = S$;
3. Each time evolution of the states in the network is a transition in T. That is, Equation (5) can be translated in to 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 the copy of V,



(a) Full BDD (b) Reduced BDD (c) Ordered reduced BDD

Figure 1: A BDD representing $f = (a \text{ OR } c) \text{ AND } b$.

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. $Image(X, T) = \{s \in S \mid \exists s' \in X$ 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. $Preimage(Y, T) = \{s \in S \mid \exists s' \in Y$ 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

In this section, we present our algorithms for detecting attractors in BNs. We propose a monolithic attractor detection algorithm for dealing with small networks and an enumerative search algorithm for large networks.F

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

```
X ≔ Z; Y ≔ ∅;
while Y ≠ X do
  | Y ≔ X; X ≔ Image(X, T);
end
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 fact in 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 $\mathcal{A} = \{A_1, \cdots, A_m\}$ be the set of attractors in Z. Algorithm 1 returns the set X of states such that $X = A_1 \cup \cdots \cup A_m$.*

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 $A_1 \cup \cdots \cup A_m \subseteq X'$.

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 $\overline{X} = X \setminus A_1 \cup \cdots \cup A_k$. 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 $\rho = s_0 s_1 s_2 \cdots$ where $s_0 = s$, $s_i \in X$ and $s_{i+1} \to s_i$ for all $i \geq 0$. Let k be the largest index in ρ such that $s_i \in \overline{X}$ for all $0 \leq i \leq k$ 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. To achieve this, 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

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

$X \coloneqq \{s\}$; $Y \coloneqq \{s\}$;
while $Y \neq \emptyset$ **do**
  $Z \coloneqq Image(Y, T)$; $Y \coloneqq Z \setminus X$; $X \coloneqq X \cup Y$;
end
$A \coloneqq Z$;
while $Z \neq \emptyset$ **do**
  $Z \coloneqq Image(Z, T)$; $Z \coloneqq Z \setminus A$; $A \coloneqq A \cup Z$;
end
return A

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

$Y \coloneqq Z$; $X \coloneqq Z$;
while $Y \neq \emptyset$ **do**
  $Y \coloneqq Preimage(Y, T)$; $Y \coloneqq Y \setminus X$; $X \coloneqq X \cup Y$;
end
return X

discovered state is a state in the attractor, from which the whole attractor can be identified, which is performed in the second **while** loop of Algorithm 2. As an example of how this algorithm works, consider the 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 **while** 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.

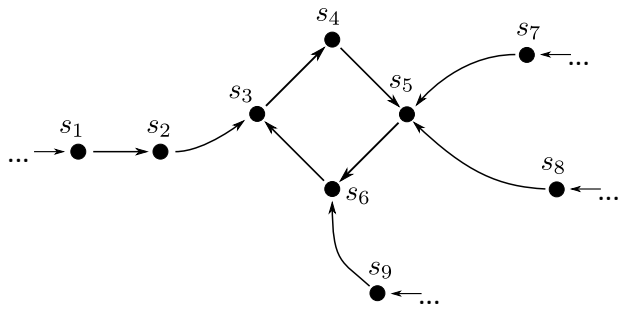


Figure 2: Part of the state space of some synchronous BN. The remaining states of the state space not shown in the figure are indicated by incoming arrows with ... to states s_1, s_7, s_8, and s_9. These arrows stand for potentially more than one incoming transition to the four states.

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.

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

$X \coloneqq Z$;
foreach $s \in Z$ **do**
  $Y \coloneqq Preimage(\{s\}, T)$; $Y \coloneqq Y \setminus Z$;
  $X \coloneqq X \cup Predecessor(Y)$;
end
return X

In Algorithm 4, the statement $Y \coloneqq Preimage(\{s\}, T)$ 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 $Y \coloneqq Y \setminus Z$ to remove s' from Y because s' will be processed separately in the **foreach** 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 **foreach** 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 the attractors in the set Z of states. Algorithm 5 starts with a randomly chosen state s in Z and computes the set X of all states reached from s and the attractor contained in X. After removing X from Z, it continues to find other attractors in the remaining states.

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 sim-

Algorithm 5: $Enumerative(Z)$: Compute all attractors in the set Z of states

$As \coloneqq \emptyset$;
while $Z \neq \emptyset$ **do**
  pick up a state $s \in Z$; $A \coloneqq Search(s)$;
  $Y \coloneqq SplitPre(A)$; $As \coloneqq As \cup A$; $Z \coloneqq Z \setminus Y$;
end
return As

ple attractor into individual states and by computing their predecessors separately can better explore the regularity of the BDDs in the system and accelerate the computation.

5. 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 quality resulting 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 with one variable in the corresponding BDD. For simplification, we treat ordering the variables of a BDD as ordering the nodes of a BN. As shown in [8, 4], variables that are topologically close within the original network should be ordered relatively close together in the BDD. In a BN, a node is closer to its parent nodes and its children nodes comparing to others. Based on this consideration, 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 with the following steps.

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, those unordered parent nodes can be ordered in an arbitrary way.
3. Repeat Step 2 until all the parent nodes of ordered nodes have been ordered.
4. If all the nodes have been ordered, the process is finished; otherwise, go to step 1.

For heuristic (II), we order the nodes with the following steps.

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, those unordered child nodes can be ordered in an arbitrary way.
3. Repeat Step 2 until all the child nodes of ordered nodes have been ordered.
4. If all the nodes have been ordered, the process is finished; otherwise, go 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 [18] 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 influential of nodes. This can be demonstrated by the experimental results in Section 6.

6. EVALUATION

We have implemented the algorithms presented in Section 4 in the model checker MCMAS [16]. In this section, we demonstrate with experiments that our algorithms can outperform the existing BDD-based attractor detection algorithm in [10] and our proposed BDD variable order heuristics can improve the performance even further.

All the experiments are conducted in 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 [19], 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.

6.1 Evaluation for small BNs

We first compare MCMAS with GenYsis for detecting attractors of small BNs. We generate BNs with nodes number 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 density: 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 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 meaning of a BN.

We name the model 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 perform the attractor detection algorithms with different BDD orders. 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 nodes 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 one (see the performance of MCMAS for 50-node BNs in Table 1 and Table 2).

6.2 Evaluation for large BNs

For large BNs, MCMAS applies the enumerative algorithm (Algorithm 5). The main difference from the algorithm of GenYsis is that MCMAS explores the predecessors of each state of a cyclic attractor separately when pruning the searching state space. We demonstrate the efficiency of this strategy 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 number, three BNs with different densities are generated. With the node number increasing, it becomes difficulty 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 compared 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 can 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 10 out of 12 of the compared cases, our proposed heuristic (I) can result in a better performance for MCMAS than the random order; while for heuristic (II), the number of better performance cases is only 7 out of 12. Besides, heuristic (I) performs better than heuristic (II) in 9 out of 12 cases. In general, both heuristics can improve the performance of MCMAS for large BNs while heuristic (I) performs better in most of the cases.

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. Increasing the density will not increase the state space of a BN, but on the other hand it exponentially increases the size of Boolean functions in the BN and the corresponding BDD representation becomes much more complex. Developing more efficient algorithms to handle large dense models is part of our future work.

7. CONCLUSION AND FUTURE WORK

In this paper, we present several strategies for improving the BDD-based attractor detection algorithm shown in [10] for synchronous BNs. Firstly, we propose to use different strategies for detecting attractors of different size BNs. For small BNs, we propose to use a monolithic algorithm which searches for attractors from the whole state space, instead of one initial state, to reduce the time for attractor detection. For large BNs, we improve the algorithm in [10] 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 detecting other attractors. Besides, we introduce two heuristics on ordering BDD variables based on the structure of BNs to further improve the attractor detection algorithm. Experimental results show that the proposed strategies can improve the BDD-based attractor detection algorithm [10] for both small and large BNs. In certain cases, the efficiency is improved by more than an order of magnitude.

As part of our future work, we will compare our algorithms with the SAT-based approaches (e.g., see [5]). We also want to extend our algorithms to deal with asynchronous BNs. Both MCMAS and GenYsis perform inefficiently for large dense BNs. One possible direction is to develop compositional approaches to deal with large dense BNs [22].

8. REFERENCES

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

[2] Bollig, B., and Wegener, I. Improving the variable ordering of OBDDs is NP-complete. *IEEE Transactions on Computers 45*, 9 (1996), 993–1002.

[3] Bryant, R. E. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys 24*, 3 (1992), 293–318.

model name	# attractor		MCMAS monolithic detection time(s)			GenYsis time(s)
	sin.	cyc.	random order	heuristic (I)	heuristic (II)	
20_*d*_01	0	1	0.13	0.09	0.12	0.65
20_*i*_02	0	4	0.04	0.04	0.06	0.32
20_*s*_03	0	3	0.04	0.03	0.03	0.08
30_*d*_04	1	0	5.25	6.81	3.80	243.56
30_*i*_05	0	10	0.59	0.45	0.48	4.62
30_*s*_06	0	3	0.22	0.22	0.22	1.02
40_*d*_07	0	2	407.38	302.89	576.26	—
40_*i*_08	4	4	23.90	24.39	25.70	64.77
40_*s*_09	2	12	2.51	1.33	3.30	6.81
50_*d*_10	0	1	270.87	360.75	391.32	—
50_*i*_11	1	0	181.83	398.96	121.86	—
50_*s*_12	1	29	1.179	1.261	1.048	7.91

Table 1: Performance of MCMAS and GenYsis for small BNs.

model name	# attractor		MCMAS split search detection time(s)			GenYsis time(s)
	sin.	cyc.	random order	heuristic (I)	heuristic (II)	
50_*d*_10	0	1	3070.11	1341.28	5398.86	—
50_*i*_11	1	0	6683.67	915.516	2801.74	—
50_*s*_12	1	29	1.02	0.87	1.16	7.91
60_*d*_13	0	5	2032.16	3117.18	1236.45	—
60_*i*_14	1	1	560.95	67.86	542.39	120.23
60_*s*_15	0	5	0.51	0.22	0.27	0.99
80_*i*_16	6	124	224.94	84.22	248.54	275.84
80_*s*_17	0	4	9.99	6.09	11.82	—
100_*s*_18	1	13	12.65	12.15	10.61	54.02
120_*s*_19	0	16	24.72	14.76	33.15	38.79
300_*s*_20	0	64	96.57	65.07	81.42	5895.26
400_*s*_21	4	700	5971.18	9533.18	5866.03	—

Table 2: Performance of MCMAS and GenYsis for large BNs.

[4] Drechsler, R. Verification of multi-valued logic networks. In *Proc. 26th Symposium on Multiple-Valued Logic* (1996), IEEE, pp. 10–15.

[5] Dubrova, E., and Teslenko, M. A SAT-based algorithm for finding attractors in synchronous Boolean networks. *IEEE/ACM Transactions on Computational Biology and Bioinformatics 8*, 5 (2011), 1393–1399.

[6] Dubrova, E., Teslenko, M., and Martinelli, A. Kauffman networks: Analysis and applications. In *Proc. 2005 IEEE/ACM International Conference on Computer-Aided Design* (2005), IEEE CS, pp. 479–484.

[7] Ferrazzi, F., Engel, F. B., Wu, E., Moseman, A. P., Kohane, I. S., Bellazzi, R., and Ramoni, M. F. Inferring cell cycle feedback regulation from gene expression data. *Journal of Biomedical Informatics 44*, 4 (2011), 565–575.

[8] Fujita, M., Fujisawa, H., and Matsunaga, Y. Variable ordering algorithms for ordered binary decision diagrams and their evaluation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 12*, 1 (1993), 6–12.

[9] Garg, A., Di Cara, A., Xenarios, I., Mendoza, L., and De Micheli, G. Synchronous versus asynchronous modeling of gene regulatory networks. *Bioinformatics 24*, 17 (2008), 1917–1925.

[10] Garg, A., Xenarios, L., Mendoza, L., and DeMicheli, G. 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* (2007), vol. 4453 of *LNCS*, Springer, pp. 62–76.

[11] Huang, S. Genomics, complexity and drug discovery: insights from Boolean network models of cellular regulation. *Pharmacogenomics 2*, 3 (2001), 203–222.

[12] Irons, D. J. Improving the efficiency of attractor cycle identification in Boolean networks. *Physica D: Nonlinear Phenomena 217*, 1 (2006), 7–21.

[13] Kauffman, S. Homeostasis and differentiation in random genetic control networks. *Nature 224* (1969), 177–178.

[14] Kauffman, S. A. Metabolic stability and epigenesis in randomly constructed genetic nets. *Journal of Theoretical Biology 22*, 3 (1969), 437–467.

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

[16] Lomuscio, A., Qu, H., and Raimondi, F. MCMAS: An open-source model checker for the verification of multi-agent systems. *International Journal on Software Tools for Technology Transfer* (2015).

[17] Luc, R. Dynamics of Boolean networks controlled by biologically meaningful functions. *Journal of Theoretical Biology 218*, 3 (2002), 331–341.

[18] Malik, S., Wang, A. R., Brayton, R. K., and Sangiovanni-Vincentelli, A. Logic verification using binary decision diagrams in a logic synthesis environment. In *Proc. IEEE International Conference on Computer-Aided Design* (1988), IEEE, pp. 6–9.

[19] Mizera, A., Pang, J., and 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* (2015), vol. 9364 of *LNCS*, Springer, pp. 214–220. Software available at `http://satoss.uni.lu/software/ASSA-PBN/`.

[20] Mushthofa, M., Torres, G., de Peer, Y. V., Marchal, K., and Cock, M. D. ASP-G: an ASP-based method for finding attractors in genetic regulatory networks. *Bioinformatics 30*, 21 (2014), 3086–3092.

[21] Needham, C. J., Manfield, I. W., Bulpitt, A. J., Gilmartin, P. M., and Westhead, D. R. From gene expression to gene regulatory networks in arabidopsis thaliana. *BMC Systems Biology 3*, 1 (2009), 85.

[22] Ravi, K., McMillan, K. L., Shiple, T. R., and Somenzi, F. Approximation and decomposition of binary decision diagrams. In *Proc. 35th Annual Design Automation Conference* (1998), ACM, pp. 445–450.

[23] Shmulevich, I., and Dougherty, E. R. *Probabilistic Boolean Networks: The Modeling and Control of Gene Regulatory Networks*. SIAM Press, 2010.

[24] Somogyi, R., and Greller, L. D. The dynamics of molecular networks: applications to therapeutic discovery. *Drug Discovery Today 6*, 24 (2001), 1267–1277.

[25] Trairatphisan, P., Mizera, A., Pang, J., Tantar, A.-A., Schneider, J., and Sauter, T. Recent development and biomedical applications of probabilistic Boolean networks. *Cell Communication and Signaling 11* (2013), 46.

[26] Zheng, D., Yang, G., Li, X., Wang, Z., Liu, F., and He, L. An efficient algorithm for computing attractors of synchronous and asynchronous Boolean networks. *PLoS ONE 8*, 4 (2013), e60593.