Determining a Singleton Attractor of a Boolean Network with Nested Canalyzing Functions

Tatsuya Akutsu[1]*, Avraham A. Melkman[2], Takeyuki Tamura[1], Masaki Yamamoto[3]

[1] Bioinformatics Center, Institute for Chemical Research, Kyoto University
Gokasho, Uji, Kyoto, 611-0011, Japan.

[2] Department of Computer Science, Ben-Gurion University of the Negev
Beer-Sheva, Israel 84105.

[3] Dept. of Informatics, School of Science and Technology, Kwansei-Gakuin University
Gakuen, Sanda, Hyogo, 669-1337, Japan.

Abstract

In this paper we study the problem of finding a singleton attractor for several biologically important subclasses of Boolean networks (BNs). The problem of finding a singleton attractor in a BN is known to be NP-hard in general. For BNs consisting of n nested canalyzing functions, we present an $O(1.799^n)$ time algorithm. The core part of this development is an $O(\min(2^{k/2} \cdot 2^{m/2}, 2^k) \cdot \text{poly}(k, m))$ time algorithm for the satisfiability problem for m nested canalyzing functions over k variables. For BNs consisting of chain functions, a subclass of nested canalyzing functions, we present an $O(1.619^n)$ time algorithm and show that the problem remains NP-hard, even though the satisfiability problem for m chain functions over k variables is solvable in polynomial time. Finally we present an $o(2^n)$ time algorithm for bounded degree BNs consisting of canalyzing functions.

Keywords: SAT, Boolean network, singleton attractor, nested canalyzing function

1 INTRODUCTION

Among various mathematical models of biological networks, the *Boolean network* (BN) is a relatively simple and well-studied one (Kauffman, 1993). The BN is a discrete model of gene regulatory networks in which each node corresponds to a gene and takes on a value of 1 or 0, meaning that the gene is or is not expressed. The value of a node at a given time instant is determined according to a regulation rule that is a Boolean function of the values of the predecessors of the node at the previous time instant. The values of nodes are updated synchronously, and the *state* of a network at a given time instant is the vector of its node values. An important characteristic of any BN is the existence of an *attractor*, among which it is customary to distinguish between a *singleton attractor* corresponding to a stable state, and a *cyclic attractor* corresponding to a sequence of states that repeats periodically.

The analysis of steady states in biological networks is an important topic in bioinformatics and systems biology, in part because of the possible correspondence between different steady states and different types of cells. We focus here on the problem of detecting a singleton attractor for a given BN.

*corresponding author

It is known that this problem (or the problem of finding an attractor of the shortest period) is NP-hard even for BNs with maximum indegree 2 (Tamura and Akutsu, 2009a). Based on an observation in (Leone et al., 2006), Tamura and Akutsu (2009a) showed that the singleton attractor detection problem for a BN with maximum indegree $K-1$ can be reduced to K-SAT, the CNF-satisfiability problem in which each clause consists of at most K literals. They also developed an $O(1.787^n)$ time algorithm for the singleton attractor detection problem for an AND/OR BN with n nodes, where an AND/OR BN is a BN in which each Boolean function is restricted to either a conjunction or a disjunction of literals (Tamura and Akutsu, 2009a). Subsequently faster algorithms for this problem were described in (Melkman et al., 2010; Tamura and Akutsu, 2009b; Yamamoto, 2010), the fastest one running in $O(1.587^n)$ time (Melkman et al., 2010). We note also that recently several heuristic algorithms have been proposed for the detection and/or enumeration of attractors (Devloo et al., 2003; Garg et al., 2008; Irons, 2006; Leone et al., 2006), and that the distribution of the numbers and lengths of attractors (Drossel et al., 2005; Samuelsson and Troein, 2003) in random BNs has attracted considerable attention.

Although the analysis of AND/OR BNs provides important insights, its immediate applicability is limited because many biological networks are not AND/OR BNs. On the other hand, all Boolean rules compiled for eukaryotes from the available literature appear to belong to the class of *canalyzing functions* (Harris et al., 2003). A canalyzing function is characterized by the property that there is at least one input such that for one of the two values of this input the value of the function is determined regardless of the values of the other variables. Furthermore, Kauffman et al. (2004) analyzed the 139 rules of Harris et al. (2003) with up to 5 inputs, and found that all but 6 of them are *nested canalyzing functions*, which are a subclass of the canalyzing functions, and that 107 of the rules are chain functions, a subclass of nested canalyzing functions introduced by Gat-Viks and Shamir (2003). Here we present three algorithms for the singleton attractor detection problem, one for BNs with nested canalyzing functions, a faster one for BNs with chain functions, and one for bounded degree BNs with canalyzing functions. In the following, we will call a nested canalyzing function an *nc-function* and a network with nc-functions an *nc-network*.

Our algorithms adopt a strategy similar to the one used in the algorithms for AND/OR BNs by (Melkman et al., 2010; Tamura and Akutsu, 2009a,b; Yamamoto, 2010). Starting from the observation that there is a family of simple BNs that is in one-to-one correspondence with the family of SAT (satisfiability) formulas, the common strategy of these algorithms is to reduce a general network to a SAT-network. A singleton attractor for the latter network is then found by the faster of two algorithms, one that exhaustively considers all possible assignments to the k variables of the SAT formula, and the other an algorithm whose running time depends only on the number m of clauses and not on k (Hirsch, 2000; Yamamoto, 2005). In order to apply the same strategy to nc-networks we define *nc-SAT*, the analogue of SAT for nc-functions, and develop an $O^*(\min(2^{(k/2)} \cdot 2^{(m/2)}, 2^k))$ time algorithm for solving nc-SAT with m nc-functions over k variables, which is of independent interest.[1] Though this algorithm for nc-SAT is obtained by combination of the standard branch-and-bound (Hirsch, 2000; Yamamoto, 2005) and maximum matching techniques (Tovey, 1984) for SAT, to our knowledge, it is the first algorithm for nc-SAT with nontrivial time complexity. This algorithm is then used as a subroutine, in an algorithm for general BNs with nc-functions that runs in $O(1.799^n)$ time.[2] The analysis of the main algorithm is done in a similar but simpler way as in Melkman et al. (2010). Interestingly, Goles and Salinas (2010) describe a polynomial-time algorithm for finding a singleton attractor in BNs having

[1]$O^*(f(k,m))$ (resp. $O^*(f(n))$) means $O(f(k,m) \cdot \mathrm{poly}(k,m))$ (resp. $O(f(n) \cdot \mathrm{poly}(n))$).

[2]The algorithm works even if there exist nodes with non-nested canalyzing functions. If there are k such nodes it runs in $O^*(2^k \cdot 1.799^{n-k})$ time, assuming that the value of each Boolean function can be computed in polynomial time.

the property that each cycle has an even number of negated edges. The nc-SAT network is a simple network without this property: each cycle in this network consists of a single negated edge.

In contrast, we show in Section 5 that SAT for chain functions can be solved in polynomial time. Nevertheless, the singleton attractor detection problem for BNs consisting of chain functions remains NP-hard. For this type of networks, we present an algorithm that runs in $O(1.619^n)$ time. We also present an $o(2^n)$ time algorithm for bounded degree BNs with canalyzing functions.

The organization of the paper is as follows. In Section 2, we define the problem and restricted Boolean functions, and give some basic tools used in the latter sections. Next, we present algorithms for nc-SAT and the singleton attractor detection problem for nc-networks in Sections 3 and 4, respectively. Next we present algorithms for BNs with chain functions and bounded degree BNs with canalyzing functions in Sections 5 and 6, respectively. The final section discusses future work. In describing the algorithms we use pseudocode for the core procedures and texts for others.

2 PRELIMINARIES

2.1 Boolean network

A *Boolean network* $N(V,F)$ consists of a set $V = \{v_1, \ldots, v_n\}$ of nodes and a corresponding set $F = \{f_v : v \in V\}$ of *Boolean functions*, where a Boolean function $f_v(v_{i_1}, \ldots, v_{i_k})$ with inputs from specified nodes $v_{i_1}, \ldots, v_{i_k}$ is assigned to each node v. We use $IN(v)$ to denote the set of input nodes $v_{i_1}, \ldots, v_{i_k}$ to v. Each node takes either 0 or 1 at each discrete time t. Let $v(t) \in \{0,1\}$ represent the value of a node v at time t, and denote by $\langle v(t) : v \in V \rangle$ the state of the network at time t. Then, the state of node v at time $t+1$ is determined by a *regulation rule* given as a Boolean function

$$v(t+1) = f_v(v_{i_1}(t), \ldots, v_{i_{k_i}}(t)).$$

We also write

$$v(t+1) = f_v(\langle v_1(t), \ldots, v_n(t) \rangle).$$

to denote this regulation rule. The network topology of a BN is represented by the directed graph $G(V,E)$ with the set of edges E defined by

$$E = \{(v_{i_j}, v_i) | v_{i_j} \in IN(v_i)\}.$$

Thus an edge from v_{i_j} to v means that v_{i_j} directly affects expression of v. The number of incoming edges to v, the number of outgoing edges, and the number of edges connecting to v in $G(V,E)$ are called the *indegree*, *outdegree*, and *degree* of v, respectively.

The values of all nodes are updated simultaneously according to the regulation rules. The initial assignment of values $\langle v(1) : v \in V \rangle$ uniquely determines the state of the network at any time in the future. An initial state is called an *attractor with period p* if $v(1) = v(p+1)$ for all $v \in V$. An attractor with period $p = 1$ is called a *singleton attractor*.

Important note: Since we are only interested in singleton attractors in this paper, we take the liberty of hiding the dependency of the value of v on t, and identifying the value of a node v with the value of the Boolean variable v; thus, for example, assigning the value 1 to the literal $\overline{v}$ means setting $v(t) \equiv 0$.

Example. A BN with regulation rules $v_1(t+1) = \overline{v_2(t)} \vee v_3(t)$, $v_2(t+1) = v_1(t) \vee (v_2(t) \wedge \overline{v_3(t)})$ and $v_3(t+1) = v_1(t) \wedge \overline{v_2(t)}$ is shown in Fig. 1 (A), and a diagram showing transition of the states of this BN is given in Fig. 1 (B). $\langle 0, 1, 0 \rangle$ is a singleton attractor.

2.2 Nested canalyzing function and chain function

In this subsection, we give the definitions of a *canalyzing function* first introduced in (Szallasi and Liang, 1998) and (Kauffman, 1993), a *nested canalyzing function* introduced in (Kauffman et al., 2003), and a *chain function* introduced in Gat-Viks and Shamir (2003).

Definition 1 (Kauffman, 1993) *A Boolean function f over $\langle v_1, ..., v_k\rangle$ is* canalyzing *if there is an $i : 1 \leq i \leq k$ and an $a \in \{0, 1\}$ such that $f(v_1, ..., v_{i-1}, a, v_{i+1}, ..., v_k)$ is constant, independent of the other inputs. The value a is called the* canalyzing value*, and the value of f given a is called the* canalyzed value*.*

Instead of the original definition of a *nested canalyzing function* (Kauffman et al., 2003) we will use an alternate definition proved to be equivalent to it in Lemma 2.6 of Jarrah et al. (2007).

Definition 2 *A Boolean function is* nested canalyzing *over $\langle v_1, ..., v_k\rangle$, if and only if it can be represented as*

$$f_v = \ell_1 \vee \cdots \vee \ell_{k_1-1} \vee (\ell_{k_1} \wedge \cdots \wedge \ell_{k_2-1} \wedge (\ell_{k_2} \vee \cdots \vee \ell_{k_3-1} \vee (\cdots))),$$

where $\ell_i \in \{v_1, \bar{v}_1, v_2, \bar{v}_2, \ldots, \bar{v}_n\}$, and $1 \leq k_1 < k_2 < \cdots$.

We will call the first clause in the representation, $\ell_1 \vee \cdots \vee \ell_{k_1-1}$, the *initial clause* of the nc-function f. If $k_1 = 1$ the initial clause is empty. We assume in this paper that nested canalyzing functions are given in the above form, where each canalzying function given in the original form can be directly transformed into one in the above form in polynomial time.

We will assume also that each nc-function in the network contains at most one appearance of any variable, since upon reaching an inner parenthesis the values of all preceding variables are known. For example, for the nc-function $f = a \vee b \vee (\overline{a} \wedge c \wedge d)$, the value of $\overline{a}$ can be assumed to be 1, since the value $a = 1$ satisfies the initial clause. Thus f can be simplified to $f = a \vee b \vee (c \wedge d)$.

A *chain function* is a special case of nested canalyzing functions as defined by (Gat-Viks and Shamir, 2003).

Definition 3 *Let $\langle v_1, ..., v_k\rangle$ and $\langle c_1, ..., c_k\rangle$ be two ordered sets of Boolean variables. The Boolean function f is a* chain function *over $\langle v_1, ..., v_k\rangle$ with control pattern $\langle c_1, ..., c_k\rangle$ if it can be represented as $f(v_1, ..., v_k) = infl_0$, where for each $1 \leq i \leq k$,*

$$infl_{i-1} = \begin{cases} v_i \wedge infl_i & if\ c_i = 0 \\ \overline{v_i \wedge infl_i} & if\ c_i = 1 \end{cases}$$

and $infl_k = 1$.

A function f is a chain-function over $\{v_1, \ldots, v_n\}$ if there is a renumbering of the variables and there is a control pattern so that f is a chain function over $\langle v_1, ..., v_k\rangle$ with control pattern $\langle c_1, ..., c_k\rangle$.

We will use instead the following alternate definition, whose equivalence can be proved in a straightforward way by induction.

Definition 3': *f is a chain function over $\langle v_1, ..., v_k\rangle$ with control pattern $c_2 = \cdots = c_{k_1-1} = 0, c_{k_1} = 1, c_{k_1+1} = \cdots = c_{k_2-1} = 0, \cdots$ if and only if it has the form of either one of the following*

$$\begin{aligned} f &= \overline{v_1} \vee \cdots \vee \overline{v_{k_1-1}} \vee (v_{k_1} \wedge \cdots \wedge v_{k_2-1} \wedge (\overline{v_{k_2}} \vee \cdots \vee \overline{v_{k_3-1}} \vee (\cdots))), \\ f &= v_1 \wedge \cdots \wedge v_{k_1-1} \wedge (\overline{v_{k_1}} \vee \cdots \vee \overline{v_{k_2-1}} \vee (v_{k_2} \wedge \cdots \wedge v_{k_3-1} \wedge (\cdots))), \end{aligned}$$

where the former and latter ones correspond to $c_1 = 1$ and $c_1 = 0$, respectively.

As in the case of nested canalyzing functions, we assume in this paper that chain functions are given in this form. It is to be noted that if $k_1 = k_2 = k_3 = \cdots$, the former and latter functions simplify to disjunction of negated variables and conjunction of variables, respectively.

2.3 Constraint nodes and variable nodes

Definition 4 *A* constraint node *in an nc-network is a node v whose nc-function has the literal $\overline{v}$ in its initial clause. All other nodes in the network are* variable nodes*. A* source node *is a variable node v with nc-function $f_v = v$. An nc-network is called an* nc-SAT-network *if it only contains source nodes and constraint nodes.*

Observe that in a singleton attractor state a constraint node necessarily has the value 1, since the value of $v(1) = 0$ does not satisfy the equation $v(1) = v(2)$. For the purpose of determining a singleton attractor it can be assumed that a constraint node has only in-edges, since its value can be propagated along each of its out-edges before starting the determination. An nc-SAT-network is therefore a bipartite network, one part containing the source nodes and the other part containing the constraint nodes.

Notation: We will denote variable nodes by $u, v, ..$, and constraint nodes by $a, b, c, ...$ Unless otherwise stated, we also denote a set of variable nodes and a set of constraint nodes by U and C, respectively. Usually the total number of nodes will be denoted by n, with the number of variable and constraint nodes denoted by k and m.

Our algorithms search for a singleton attractor state by splitting all possible assignments into ones in which a certain node v has the value 1 and ones in which its value is 0. Since the value of a node is determined by its nc-function, this means in both cases that a constraint is imposed on the predecessors of v. We show here how to incorporate such a constraint into the network by generating an appropriate constraint node to replace the variable node whose value is fixed. In the algorithms to follow, we will call this procedure **Convert2Constraint**(ℓ) where ℓ is a literal. The purpose of the procedure is to impose the constraint $\ell = 1$ on the variable node v associated with the literal ℓ.

If $\ell = v$, the case of a 1-constraint, the method generates a constraint node w with nc-function $f_w = \overline{w} \vee f_v$. Here every appearance of v or $\overline{v}$ in f_v and the nc-functions of the successors of v is replaced by w or $\overline{w}$ respectively. Note that this construction also addresses the case $f_v = v \vee g_v$, resulting in $f_w \equiv 1$, i.e. no constraint is imposed.

If $\ell = \overline{v}$, the 0-constraint case, the idea is to impose a 1-constraint on $\overline{v}$. Thus a constraint node w is generated with nc-function $f_w = \overline{w} \vee \overline{f_v}$. Here every appearance of v or $\overline{v}$ in $\overline{f_v}$ and the nc-functions of the successors of v is replaced by $\overline{w}$ or w respectively. In the particular case that $f_v = v \vee g_v$ it follows that $f_w = \overline{w} \vee (w \wedge \overline{g_v}) = \overline{w} \vee \overline{g_v}$.

2.4 Canonical network and reduced network

In this subsection, we introduce some tools used in our main algorithm. First, we introduce a *canonical form* of an nc-network. Recall that it can be assumed that any variable makes at most one appearance in an nc-function.

Definition 5 *We say that an nc-network is in a* canonical form *if each nc-function contains at most one appearance of any variable, has non-empty initial clause, $k_1 > 1$, and the initial clause of each constraint node c contains at least one literal other than $\overline{c}$.*

Then, we introduce the **Reduce**(N, L) procedure that simplifies a given canonical nc-network N under an initial partial assignment given as a list of literals whose values are 1, where it is almost the same as one in (Melkman et al., 2010). Given a network N and a partial assignment L (i.e., a list of literals whose values are 1), **Reduce**(N, L) gives a reduced network and an expanded partial assignment, which we call the *accompanying assignment.*

```
Reduce$(N, L)$ {$N$ is a canonical network, $L$ is a list of literals; returns a reduced canonical network
    and an accompanying partial assignment $A$};
begin
initialize $A$ to $\emptyset$;
while $L$ is not empty do
    extract a literal $\ell$ from $L$ and insert it into $A$;
    for each successor node $v$ of the node $u$ associated with $\ell$ do
        in $f_v$ replace $\ell$ by 1, or $\overline{\ell}$ by 0, and simplify $f_v$;
        if $f_v = 1$ then remove $v$ from $N$ and add $v$ to $L$,
        else if $f_v = 0$ then remove $v$ from $N$ and add $\overline{v}$ to $L$,
        else if the initial clause of $f_v$ has the value 0 then replace $v$ by a node $w$ with $f_w = \overline{f_v}$
            and replace $v$ by $\overline{w}$ in the nc-function of each successor of $v$; {comment: keeps network
            canonical}
return $(N, A)$;
end.
```

Simplifying f_v means

- deleting a value of 0 from an or-clause, and a value of 1 from an and-clause;[3]
- replacing an entire or-clause by 1 if one of its literals is 1, and replacing an entire and-clause by 0 if one of its literals is 0.

Thus, for example, $f_v = \ell_1 \vee \cdots \vee \ell_{k_1-1} \vee (\ell_{k_1} \wedge \cdots \wedge \ell_{k_2-1} \wedge (\ell_{k_2} \vee \cdots \vee \ell_{k_3-1} \vee (\cdots)))$, simplifies to $f_v = \ell_1 \vee \cdots \vee \ell_{k_1-1}$ if $\ell_{k_1} = 0$.

Clearly, there is a singleton attractor A' in the reduced network N', if and only if there is a singleton attractor $A' \cup L$ in the network N. It is also clear that the time requirement of **Reduce** is only polynomial.

Example. Fig. 2 shows an example explaining **Convert2Constraint** and **Reduce**. Let the network shown in Fig. 2 (A) be a part of N with $f_v = x_1 \vee x_2$, $f_{y_1} = \overline{v} \vee (x_3 \wedge \overline{x_4})$, $f_{y_2} = v \vee x_4$, and $f_{y_3} = y_1 \vee \overline{y_2}$.

[3]We call the part of $(\ell_h \wedge \cdots \wedge \ell_k \wedge (\cdots))$ (resp. $(\ell_h \vee \cdots \vee \ell_k \vee (\cdots))$) of an nc-function an *and-clause* (resp. *or-clause*) though it is not a clause.

In this example, **Convert2Constraint**(v) is first applied to N, then the network N' shown in Fig. 2 (B) is obtained, where $f_w = \overline{w} \vee x_1 \vee x_2$, $f_{y_1} = \overline{w} \vee (x_3 \wedge \overline{x_4})$, and $f_{y_2} = w \vee x_4$. Next, **Reduce**$(N', \{w\})$ is applied to N'. Then, f_{y_1} is simplified into $f_{y_1} = x_3 \wedge \overline{x_4}$. Since f_{y_1} is not canonical, y_1 is replaced by $y_1' = \overline{y_1}$ and we have $f_{y_1'} = \overline{x_3} \vee x_4$ and $f_{y_3} = \overline{y_1'} \vee \overline{y_2}$. Then, f_{y_2} is simplified into $f_{y_2} = 1$ and thus y_2 is removed from N' and is added to L. Then, f_{y_3} is simplified into $f_{y_3} = \overline{y_1'}$.

Using **Reduce**, we have the following.

Lemma 1 *A general nc-network can be converted into a canonical form in polynomial time.*

Proof: Suppose that there is a variable node v in the network whose nc-function has an empty initial clause, $k_1 = 1$, i.e. $f_v = \ell_1 \wedge \cdots \wedge \ell_{k_2-1} \wedge (\ell_{k_2} \vee \cdots \vee \ell_{k_3-1} \vee (\cdots)))$. Construct an equivalent network in which the node v is replaced by the node w, with nc-function $f_w = \overline{f_v}$, and for each non-negated (negated) edge from v to x there is a negated (non-negated) edge from w to x. Note that the initial clause of f_w is $\overline{\ell_1} \vee \cdots \vee \overline{\ell_{k_2-1}}$, which is non-empty since $k_2 > 1$. In effect, $w = \overline{v}$.

Suppose now that there is a constraint node c with initial clause $\overline{c}$, $f_c = \overline{c} \vee (\ell_2 \wedge \cdots \wedge \ell_{k_2-1} \wedge (\ell_{k_2} \vee \cdots \vee \ell_{k_3-1} \vee (\ell_{k_3} \wedge \cdots)))$. It follows that c can have the value 1, its only possible value in a singleton attractor, only if $\ell_2 = \cdots = \ell_{k_2-1} = 1$, and if that holds $f_c = \overline{c} \vee \ell_{k_2} \vee \cdots \vee \ell_{k_3-1} \vee (\ell_{k_3} \wedge \cdots)$. Thus upon generation of constraint nodes for $\ell_2 = \cdots = \ell_{k_2-1} = 1$ and constructing a reduced network with these assignments, c has the requisite form in the reduced network.

These steps can be repeated until the network has the canonical form. □

3 DETECTION OF A SINGLETON ATTRACTOR IN AN NC-SAT-NETWORK

An nc-SAT-network has a singleton attractor if and only if there is an assignment to the variables $v_1, \cdots, v_k$ that satisfies a set of constraints $\{g_c = 1 : c \in C\}$, with each g_c a nested canalyzing function over $v_1, \cdots, v_k$. The nc-SAT-network is a bipartite network, with one part containing source nodes (nodes s with $f_s = s$), corresponding to the variables, and the other part containing constraint nodes (nodes c whose initial clause contains the literal $\overline{c}$), corresponding to the constraints so that $f_c = \overline{c} \vee g_c$, meaning g_c must have value 1. We assume that the network has been made canonical, and call N a *canonical nc-SAT-network* if an nc-SAT-network N is in a canonical form.

3.1 Algorithm for an nc-SAT-Network

The following is a pseudo code of the algorithm for detection of a singleton attractor in an nc-SAT-network (i.e., for solving SAT for nc-functions).

```
NC-SAT-Attractor($N$) {$N$ is a canonical nc-SAT network; returns $(b, A)$, with $b$ = true if a
    singleton attractor $A$ exists, $b$ = false otherwise };
begin
if $N = \emptyset$ then return $(true, \emptyset)$;
if there is a node $c$ with $f_c = \overline{c}$ then return $(false, \emptyset)$;
if there is a constraint node $c$ with an initial clause of length 1, $f_c = \overline{c} \vee (\ell_2 \wedge \cdots \wedge \ell_{k_2-1} \wedge (\ell_{k_2} \vee \cdots))$,
    then
```

```
        let $N'$ be obtained by assigning $\ell_2 = \cdots = \ell_{k_2-1} = 1$ to $N$;
        **if NC-SAT-Attractor**($N'$) returns $(true, A')$ **then return** $(true, A' \cup \{\ell_2, \ldots, \ell_{k_2-1}\})$;
        **else return** $(false, \emptyset)$;

**if** there is a node $c$ with initial clause of length 2, $f_c = \overline{c} \vee \ell_2 \vee (\ell_3 \wedge \cdots \wedge \ell_{k_2-1} \wedge (\ell_{k_2} \vee \cdots))$ (**Case 1**), **then**
        let $N_1$ be obtained by assigning $\ell_2 = 1$ to $N$;
        let $N_0$ be obtained by assigning $\ell_2 = 0$, $\ell_3 = \cdots = \ell_{k_2-1} = 1$ to $N$;
        **if NC-SAT-Attractor**($N_0$) returns $(true, A')$ **then return** $(true, A' \cup \{\ell_2\})$;
        **else if NC-SAT-Attractor**($N_1$) returns $(true, A')$ **then return** $(true, A' \cup \{\overline{\ell_2}, \ell_3, \ldots, \ell_{k_2-1}\})$;
        **else return** $(false, \emptyset)$;

**if** there is a literal $\ell \neq \overline{c}$ which appears in the initial clauses of at least two constraint nodes, and which appears negated in the initial clause of at least one other constraint node (**Case 2**), **then**
        let $N_0$ be obtained by assigning $\ell = 0$ to $N$;
        let $N_1$ be obtained by assigning $\ell = 1$ to $N$;
        **if NC-SAT-Attractor**($N_0$) returns $(true, A')$ **then return** $(true, A' \cup \{\overline{\ell}\})$;
        **else if NC-SAT-Attractor**($N_1$) returns $(true, A')$ **then return** $(true, A' \cup \{\ell\})$;
        **else return** $(false, \emptyset)$;

**else** {**(Case 3)**} **Match**($N$);

**end.**
```

Next we detail the algorithm **Match(**N**)**.

```
**Match(**$N$**)** {*precondition: $N$ is a canonical nc-SAT-network, in which the initial clause of each constraint node is of length at least 3, and each literal that appears negated in one initial clause and non-negated in another appears in no other initial clause*}; {*returns a singleton attractor $A$*}

**begin**

create a partial assignment $A$ in which each literal that appears in more than two initial clauses is set to 1;

let $C$ be the set of initial clauses that remain unsatisfied after these assignments, and let $U$ be the set of remaining unassigned variables;

construct a bipartite graph with a node $u$ for each $u \in U$, a node $c$ for each $c \in C$, and an edge $(u, c)$ whenever a variable $u$ appears, negated or not, in an initial clause $c$;

find a matching saturating $C$: $\{(u_c, c) : c \in C\}$ ;

for each $c \in C$ add $u_c$ to $A$ if $u_c$ appears non-negated in $c$ and add $\overline{u_c}$ otherwise;
```

```
  return $(true, A)$;
  end.
```

In order to make the algorithm more readable, we have used phrases such as "let N' be obtained by assigning $\ell_2 = 1$ to N;" What is meant by this is to first execute **Convert2Constraint**(ℓ_2) and $(N', A') \leftarrow$**Reduce**$(N, \{\ell_2\})$.

Example.
In Fig. 3, $u, v, w, \ldots$ represent variable nodes and $a, b, c, \ldots$ represent constraint nodes.

In Fig. 3 (A), BN contains regulation rules of

$$\begin{aligned} f_a &= \overline{a} \vee \overline{u} \vee v \vee w \vee (\cdots), \\ f_b &= \overline{b} \vee v \vee w \vee (u \wedge \cdots), \\ f_c &= \overline{c} \vee u \vee (v \wedge \overline{w} \wedge \cdots). \end{aligned}$$

Then, the lengths of initial clauses of a, b, c are 4,3,2 respectively. Note that self loops are also counted as literals of initial clauses. Node c satisfies the condition of Case 1. If $u = 1$ is assigned, f_c is satisfied. Otherwise, $v = 1$ and $w = 0$ must hold.

In Fig. 3 (B), BN contains regulation rules of

$$\begin{aligned} f_a &= \overline{a} \vee u \vee v \vee (w \wedge \cdots), \\ f_b &= \overline{b} \vee u \vee v \vee (\overline{w} \wedge \cdots), \\ f_c &= \overline{c} \vee \overline{u} \vee w \vee (\overline{v} \wedge \cdots). \end{aligned}$$

Literal u satisfies the condition of Case 2. If $u = 1$ is assigned, f_a and f_b are satisfied. Otherwise, f_c is satisfied.

In Fig. 3 (C), BN contains regulation rules of

$$\begin{aligned} f_a &= \overline{a} \vee \overline{u} \vee v \vee (\overline{w} \wedge \cdots), \\ f_b &= \overline{b} \vee \overline{v} \vee \overline{w} \vee (u \wedge \cdots), \\ f_c &= \overline{c} \vee u \vee w \vee (v \wedge \cdots). \end{aligned}$$

There exists two matchings saturating $\{a, b, c\}$; $\{(u, a), (v, b), (w, c)\}$ and $\{(v, a), (w, b), (u, c)\}$. The former leads to an assignment of $(u = 0, v = 0, w = 1)$ whereas the latter leads to an assignment of $(u = 1, v = 1, w = 0)$.

3.2 Correctness and running time of the algorithm

We prove first that algorithm **Match**(N) always finds a singleton attractor if the precondition of the algorithm holds.

Proposition 1 *Algorithm* **Match** *returns an assignment that is a singleton attractor of N, provided N satisfies the precondition of the algorithm. The running time of the algorithm is polynomial.*

Proof: The precondition ensures that if a literal appears in more than two initial clauses, then its negation does not appear in any initial clause. Therefore the initial partial assignment A constructed by the algorithm is unequivocal.

It only remains to show that a matching saturating C is always found. For that purpose, we prove that Hall's condition always holds: for every $C' \subseteq C$ the set of its neighbors $NB(C') = \{u : \textit{u is a neighbor of some } c \in C'\}$ has size no less than $|C'|$. Indeed, since by the precondition each remaining initial clause is of length at least 3 (length 2 excluding $\overline{c}$), the number of edges out of C' is no less than $2|C'|$. On the other hand, the number of edges is at most $2|NB(C')|$ since each variable appears in at most two initial clauses. Thus $|NB(C')| \geq |C'|$.

It is known that a maximum matching (i.e., matching saturating C) can be found in polynomial time. □

Theorem 1 *Algorithm* **NC-SAT-Attractor**(N) *finds a singleton attractor of a canonical nc-SAT-network if and only if there exists one. Its running time is* $O^*(\min(2^k, 2^{k/2}2^{m/2}))$*, where* k *is the number of source nodes and* m *is the number of constraint nodes .*

Proof: For the correctness, it is sufficient to verify that every singleton attractor is among the states of the network considered by the algorithm. Recall that each initial clause in a canonical nc-SAT-network contains at least one literal associated with a variable, and observe that the algorithm distinguishes between 3 cases that together cover all possible forms of canonical nc-SAT-networks:
Case 1: There exists a constraint node c whose initial clause contains a single literal associated with a variable.
Case 2: There exists a literal that appears in at least 2 initial clauses and appears negated in at least one other initial clause.
Case 3: All initial clauses contain at least 2 literals associated with a variable, and each such literal that appears negated in one initial clause and non-negated in another appears in no other initial clauses.

Only in case 1 are some states immediately excluded from consideration, namely those states for which $\ell_2 = \ell_3 = 0$. However, for such states necessarily $f_c = \overline{c} \vee \ell_2 \vee (\ell_3 \wedge \cdots) = 0$, so that they cannot be singleton attractors.

In order to analyze the running time, let us define $f(k,m)$ to be the worst case running time of the algorithm for a canonical nc-SAT-network with k variable nodes and m constraint nodes. Clearly, we have $f(k,0) = f(k,1) = f(k,2) = f(1,m) = f(2,m) = O^*(1)$. Then the three cases yield the following recurrence inequalities for $f(k,m)$.
Case 1: $f(k,m) \leq f(k-1,m-1) + f(k-2,m) + O^*(1),\ k \geq 2.$
Case 2: $f(k,m) \leq f(k-1,m-2) + f(k-1,m-1) + O^*(1),\ m \geq 3.$
Case 3: $f(k,m) = O^*(1).$

Therefore, we have

$$f(k,m) \leq \max\{f(k-1,m-2) + f(k-1,m-1), f(k-1,m-1) + f(k-2,m)\} + O^*(1).$$

From this inequality it follows by induction that

$$f(k,m) \leq B \min\{2^k, 2^{k/2}2^{m/2}\} + O^*(1),$$

for some constant B. Namely, for the induction step it suffices to observe that

$$\begin{aligned} f(k-1,m-2) + f(k-1,m-1) &\leq B\ \min\{2^{(k+m)/2}(2^{-3/2} + 2^{-1}), 2^{k-1} + 2^{k-1}\} + O^*(1) \\ &\leq B\ \min\{2^{(k+m)/2}, 2^k\} + O^*(1), \end{aligned}$$

and

$$\begin{aligned} f(k-1,m-1)+f(k-2,m) &\leq B\ \min\{2^{(k+m-2)/2}(1+1), 2^{k-1}+2^{k-2}\}+O^*(1) \\ &\leq B\ \min\{2^{(k+m)/2}, 2^{k}\}+O^*(1). \end{aligned}$$

□

More details concerning the running time are given in Appendix.

4 DETECTION OF A SINGLETON ATTRACTOR IN A GENERAL NC-NETWORK

The basic idea of our main algorithm for nc-networks is similar to the one in (Melkman et al., 2010). It reduces a given nc-network to an nc-SAT-network, by repeatedly converting a variable node that appears in the initial clause of another variable node into a constraint node. Thus each assignment induces a new network obtained by eliminating the nodes whose values are implied by the assignment, and adding constraint nodes representing the induced requirements, if any. Once a reduced network has no edges between variable nodes it is an nc-SAT-network, for which an attractor is found using the algorithm of Section 3.

4.1 Algorithm and its analysis

Here we give a pseudo code of our main algorithm for nc-networks.

```
NC-Attractor($N$) {$N$ is a canonical nc-network with $k$ variable nodes and $m$ constraint nodes;}
    {returns a singleton attractor $A$ if it exists};
begin
if there is a variable node $u$ which appears, negated or not, in the initial clause of another variable
    node $v$ then
      Convert2Constraint($\overline{u}$);
      $(N_0, A_0) \leftarrow$ Reduce($N, \{\overline{u}\}$);
      if NC-Attractor($N_0$) returns a singleton attractor $A'$ then return (true,$A' \cup A_0$);
      Convert2Constraint($u$);
      $(N_1, A_1) \leftarrow$ Reduce($N, \{u\}$);
      {comment: at least one of $N_0$ and $N_1$ has at most $k-2$ variable nodes, and the other at most
          $k-1$; each network has no more than $n$ nodes in total};
      if NC-Attractor($N_1$) returns a singleton attractor $A'$ then return (true,$A' \cup A_1$);
      return (false,$\emptyset$);
else {comment: $N$ is an nc-SAT-network }
      if NC-SAT-Attractor($N$) returns a singleton attractor $A'$ then return (true, $A'$)
      else return (false,$\emptyset$);
```

end.

As mentioned before, the basic strategy of this algorithm is similar to that of (Melkman et al., 2010). The main differences are the following.

- In the present case one of the 0/1 assignments to the variable node v is guaranteed to remove at least two variable nodes and the other removes at least one. In the corresponding situation of (Melkman et al., 2010) either both of the 0/1 assignments to the variable node v are guaranteed to remove two variable nodes, or one removes at least 3 variable nodes and the other at least one.
- The algorithm for usual SAT is employed at the final stage of the algorithm in (Melkman et al., 2010), whereas our newly developed **nc-SAT-Attractor** is used here.

By modifying the proof in (Melkman et al., 2010) with taking into these differences account, we have the following theorem, where we provide here the detailed proof for the sake of self-containedness.

Theorem 2 *A singleton attractor of an nc-network with n nodes can be found in time $O(1.799^n)$.*

Proof: It is apparent from the pseudo-code that the number of variable nodes decreases with each recursive call. Therefore the base of the recursion, an nc-SAT-network, is always reached. **NC-SAT-Attractor** then provides the correct answer. The correctness of the recursive calls is clear: N has a singleton attractor in which $u = 0$ (or $u = 1$) if and only if N_0 (or N_1) has a singleton attractor, and there is a one-to-one correspondence between the assignments.

For the purpose of estimating the time required by the algorithm, we conceptually break its operation up into two phases. The first phase comprises those invocations with a number of variable nodes that is at least $n/2$, and the second phase comprises the remaining invocations. The first phase therefore ends with two kinds of base cases: invocations in which **NC-SAT-Attractor** is called before $n/2$ variable nodes have been removed, and invocations in which at least $n/2$ variable nodes have been assigned values.

An invocation of **NC-Attractor** for a network with k variable nodes makes at most two recursive calls, and at least one of N_0 and N_1 has at most $k-2$ variable nodes, and the other at most $k-1$. To see why this is so observe that u is chosen because it appears in the initial clause of v; if it appears negated then assigning $u = 0$ also causes v to be set to 1; thus N_0 has at most $k-2$ variable nodes. Similarly, if u appears non-negated in the initial clause of v then N_1 has at most $k-2$ variable nodes.

Thus the number of base cases reached when aiming for the removal of q variable nodes is upper bounded by the Fibonacci function $F(q)$, satisfying the equation $F(q) \leq F(q-1) + F(q-2)$, $F(0) = F(1) = 1$. Hence $F(q)$ is $O(\lambda^q)$ with $\lambda = 1.6181 > x$, where x is the solution to $x^2 = x + 1$. It follows that the number of base cases reached in the first phase is $O(\lambda^{n/2})$.

Consider now the time required for each kind of base case. Since the total number of nodes never increases in any recursive call, Theorem 1 ensures that whenever **NC-SAT-Attractor** is used the time it requires is $O^*(2^{n/2})$. If, on the other hand, a base case is reached in which at least $n/2$ variable nodes have been assigned values, so that at most $n/2$ variables have yet to be assigned value, then one can simply go over all possible assignments, at most $2^{n/2}$ in number, and in polynomial time check whether the assignment is a singleton attractor. The time needed for such a base case is therefore also $O^*(2^{n/2})$. Therefore, the total computation time is $O^*(2^{n/2}) \cdot O(\lambda^{n/2}) < O(1.799^n)$.

The pseudo-code of the algorithm does not explicitly use the second phase as just described. Instead, as long as there is a variable node u which appears in the initial clause of another variable node the algorithm continues as before, in effect using the constraints embodied by the edges to complete the partial assignments more efficiently than the brute force method. It is easily proved by induction that this computation is at least as efficient, requiring no more than $O^*(2^{n/2})$ time, starting from an invocation of **NC-SAT-Attractor** with k variable nodes, $k \leq n/2$ which according to Theorem 1 requires no more than $O^*(2^k)$ time. □

5 A FASTER ALGORITHM FOR CHAIN FUNCTIONS

In this section, we describe a faster algorithm for BNs consisting of chain functions. It uses the somewhat surprising observation that the analogue of SAT for chain functions is solvable in polynomial time. For brevity we refer to a network with chain functions as a *cf-network*, and to the analogue for chain functions of an nc-SAT-network as a *cf-SAT-network*.

We will repeatedly make use of the representation of chain functions given in Prop. 2.2. It should be noted that by its definition a chain function over $v_1, v_2, ..., v_k$ is not a chain function over $\overline{v_1}, v_2, ..., v_k$. For that reason we will not make use of the canonical representation of a network in this section (i.e., we ignore the term $\overline{c}$). Consequently, a chain function has the form of either $\overline{v_1} \vee \cdots \vee \overline{v_{k_1-1}} \vee (v_{k_1} \wedge \cdots)$ or $v_1 \wedge \cdots \wedge v_{k_1-1} \wedge (v_{k_1} \vee \cdots)$.

Lemma 2 *A singleton attractor in a cf-SAT-network can be found in polynomial-time.*

Proof: The first step in finding a singleton attractor is to consider those constraint nodes having the form of $v_{i_1} \wedge \cdots \wedge v_{i_{k_1-1}} \wedge (v_{i_{k_1}} \vee \cdots)$, and to set $v_{i_1} = \cdots = v_{i_{k_1-1}} = 1$. These assignments are then substituted in all of the chain functions. This step is repeated until either one of the chain functions evaluates to 0, meaning that the network does not have a singleton attractor, or else all chain functions that do not evaluate to 1 have the form of $\overline{v_1} \vee \cdots \vee \overline{v_{k_1-1}} \vee (v_{k_1} \wedge \ldots)$. In the latter case a singleton attractor is constructed by assigning the value 0 to all unassigned variable nodes, thereby ensuring that all constraint nodes have the value 1. Clearly a singleton attractor is found if and only if one exists, since the assignments generated in the first step must also be part of any singleton attractor. □

Based on this lemma, we have the following.

Theorem 3 *A singleton attractor of a general cf-network with n nodes can be found in time $O(1.619^n)$.*

Proof: The algorithm for the detection of a singleton attractor is almost the same as **NC-Attractor**(N). One change is that when a cf-SAT-network is reached the algorithm calls the polynomial-time method. Another change that has to be made occurs in the generation of a 0- or 1-constraint for a variable node; its purpose is to maintain the property that all regulation functions are chain functions. Consider, for example, the imposition of a 1-constraint on a node v with $f_v = \overline{v_1} \vee \cdots \vee \overline{v_{k_1-1}} \vee (v_{k_1} \wedge \cdots \wedge v_{k_2-1} \wedge (\overline{v_{k_2}} \vee \cdots))$. If $k_1 > 1$ then the imposition of a 1-constraint is implemented by replacing v by w with $f_w = \overline{w} \vee f_v$. If $k_1 = 1$ then it is implemented by imposing 1-constraints on $v_{k_1}, \cdots, v_{k_2-1}$, and replacing v by w with $f_w = \overline{w} \vee \overline{v_{k_2}} \vee \cdots$. The imposition of a 0-constraint on v is again implemented by imposing a 1-constraint on $\overline{v}$.

The last change occurs in the **Reduce** method: in case the initial clause of f_v has the value 0 nothing needs to be done since we do not insist on the network having canonical form.

The running time is easily evaluated using the same ideas used in the proof of Theorem 2. Here we aim for the removal of all variable nodes. Thus the number of base cases is at most λ^n, and each such case is solved in polynomial time. $\square$

Although SAT for chain function can be solved in polynomial time, the singleton attractor detection problem for cf-networks remains NP-hard.

Theorem 4 *It is NP-hard to detect a singleton attractor in a cf-network.*

Proof: We show that the CNF-SAT problem (i.e., the standard SAT) can be reduced to the problem of deciding whether a cf-network has a singleton attractor.

Suppose then that an instance of CNF-SAT is given with k variables $x_i,\ i = 1, \cdots, k$, and m clauses $C_j,\ j = 1, \cdots, m$, $C_j = \ell_1^j \vee \ell_2^j \vee \cdots$. The corresponding network contains nodes, $u_i, v_i,\ i = 1, \cdots, k$, with regulation functions $f_{u_i} = \overline{v_i}, f_{v_i} = \overline{u_i}$, and constraint nodes $c_j,\ j = 1, \cdots, m$. The regulation function for c_j is of the form $f_{c_j} = \overline{c_j} \vee \overline{w_1^j} \vee \overline{w_2^j} \vee \cdots$, where $w_i^j = u_i^j$ if $\ell_i^j = \overline{x_i^j}$, and $w_i^j = v_i^j$ if $\ell_i^j = x_i^j$. Clearly the network is a cf-network since all regulation functions are chain functions.

To verify the correctness of the reduction observe that the regulation functions f_{u_i}, f_{v_i} ensure that in any singleton attractor of this network each pair (u_i, v_i) either has the values $(0, 1)$ or the values $(1, 0)$. Thus the assignment $x_i = u_i, \overline{x_i} = v_i$ is consistent. Furthermore, the regulation function $f_{c_j} = \overline{c_j} \vee \overline{w_1^j} \vee \overline{w_2^j} \vee \cdots$ ensures that in the singleton attractor $\overline{w_1^j} \vee \overline{w_2^j} \vee \cdots = 1$, so that the clause C_j is satisfied. $\square$

6 AN ALGORITHM FOR BOUNDED DEGREE NETWORKS WITH CANALYZING FUNCTIONS

Though we have focused on restricted forms of canalyzing functions, it is reasonable to ask whether there is an efficient algorithm for BNs with general canalyzing functions. Unfortunately, the singleton attractor detection problem with general canalyzing functions is as difficult as the one with general Boolean functions because the latter can be reduced to one with canalyzing functions as follows. Add nodes u, v with regulation functions $f_u = u, f_v = \overline{u} \vee \overline{v}$, and change the regulation function f_w of each existing node w to the canalyzing function g_w that equals 1 if $v = 0$ and equals f_w if $v = 1$. This network has a singleton attractor if and only if the original network had one, because in any singleton attractor necessarily $u = 0, v = 1$. Furthermore, since the space required to describe the regulation functions is $\Omega(2^n)$ (because there exist $\Theta(2^{2^n})$ Boolean functions with n input variables), it is not plausible to have an $o(2^n)$ time algorithm.

On the other hand, it is known that the singleton attractor detection problem for a BN with n nodes of maximum indegree K can be reduced to $(K+1)$-SAT with n variables (Tamura and Akutsu, 2009a).[4] The following theorem shows that we can do better for BNs with canalyzing functions than this simple reduction if all the indegrees and outdegrees are bounded by a constant K and each node has a canalyzing input from another node.

Theorem 5 *Let N be a BN with canalyzing functions in which each node has both indegree and outdegree bounded by a constant K and has a canalyzing input from another node. Then a singleton attractor of N can be found in $O^*([3^{1/(K+1)} \cdot (2 - 2/(K+2))^{1-2/(K+1)}]^n)$ time, which is at most $O^*(2^{g(K)n})$ where $g(K) = 1 - (\frac{\log(4/3)}{K+1} + \frac{\log e}{K+2} - \frac{2 \log e}{(K+1)(K+2)})$.*

[4]Each Boolean function can be represented in constant space if K is a constant.

Proof: Since each canalyzing function has at least one canalyzing input[5], we consider a subnetwork of a given BN in which each node has one incoming edge corresponding to a canalyzing input. In this subnetwork, we have a matching of size at least $n/(K+1)$ because the maximum outdegree is bounded by K. For each matching edge (u, v) (u is an input to v), it is enough to examine 3 assignments because u is a canalyzing input to v. We consider the following algorithm as in (Tamura and Akutsu, 2009a):

1. Examine all $3^{n/(K+1)}$ possible assignments on $n/(K+1)$ pairs,

2. For each such partial assignment, reduce the singleton attractor detection problem to $(K+1)$-SAT and solve $(K+1)$-SAT.

Since K-SAT on n variables can be deterministically solved in $O^*((2-2/(K+1))^n)$ time (Dantsin et al., 2002) and the number of non-assigned nodes after Step 1 is $n - 2n/(K+1)$, this algorithm works in time $O^*(t(K)^n)$, where

$$\begin{aligned} t(K) &= 3^{\frac{1}{K+1}} \cdot \left(2 - \frac{2}{K+2}\right)^{1-\frac{2}{K+1}} \\ &= 3^{\frac{1}{K+1}} \cdot 2^{1-\frac{2}{K+1}} \cdot \left(1 - \frac{1}{K+2}\right)^{1-\frac{2}{K+1}} \\ &\leq 3^{\frac{1}{K+1}} \cdot 2^{1-\frac{2}{K+1}} \cdot e^{-\frac{1}{K+2}(1-\frac{2}{K+1})} \\ &= 2^{\frac{\log 3}{K+1}} \cdot 2^{1-\frac{2}{K+1}} \cdot 2^{-\frac{\log e}{K+2}(1-\frac{2}{K+1})} \\ &= 2^{1-(\frac{\log(4/3)}{K+1}+\frac{\log e}{K+2}-\frac{2\log e}{(K+1)(K+2)})}. \end{aligned}$$

□

It is easy to see from the upper bound $O^*(2^{g(K)n})$ that the running time is strictly less than 2^n for any constant $K \geq 2$. By an elementary calculation, we also see that if $K > 2 + 2\sqrt{3}$ (i.e., $K \geq 6$), the running time of Theorem 7 is less than $O^*((2 - 2/(K+2))^n)$, which is obtained by first reducing to $(K+1)$-SAT and then using the best available deterministic algorithm (Dantsin et al., 2002) for SAT.

7 CONCLUSION

In this paper, we have presented $O(1.799^n)$ time and $O(1.619^n)$ time algorithms for the singleton attractor detection problem for Boolean networks consisting of nested canalyzing functions and chain functions, respectively. To our knowledge, they are the first algorithms with the non-trivial worst-case time complexity for these classes of Boolean networks. We have also presented an $o(2^n)$ time algorithm for the singleton attractor detection problem for bounded degree Boolean networks with canalyzing functions while we showed some evidence suggesting the difficulty of developing an $o(2^n)$ time algorithm unless there exists a degree constraint. These results shed light on the singleton attractor problem for special but biologically important classes of Boolean networks from an algorithmic viewpoint and might stimulate further improvements and developments.

Though we focused on detection of singleton attractors in this paper, enumeration of singleton attractors is also important. Since no algorithm with the non-trivial worst-case time complexity is known for enumeration of singleton attractors, it should be studied. It is also important to study detection or enumeration of cyclic attractors (i.e., attractors with period more than one). Again, no

[5]We can ignore nodes without input edges.

algorithm with the non-trivial worst-case time complexity is known for this problem though there exist some results on the average case time complexity for detection of cyclic attractors with short periods (Zhang et al., 2007) Therefore, development of such algorithms is left as an open problem.

ACKNOWLEDGMENTS

Tatsuya Akutsu was partially supported by Grant-in-Aid #22650045 from MEXT, Japan.

DISCLOSURE STATEMENT

No competing financial interests exist.

References

Dantsin, E., Goerdt, A., Hirsch, E.A., Kannan, R., Kleinberg, J., Papadimitriou, C., Raghavan, P., and Schöning, U. 2002. A deterministic $(2-2/(k+1))^n$ algorithm for k-SAT based on local search. *Theoretical Computer Science* 289, 69–83.

Devloo, V., Hansen, P., and Labbé, M. 2003. Identification of all steady states in large networks by logical analysis. *Bulletin of Mathematical Biology* 65, 1025–1051.

Drossel, B., Mihaljev, T., and Greil, F. 2005. Number and length of attractors in a critical Kauffman model with connectivity one. *Physical Review Letters* 94, 088701.

Garg, A., DiCara, A., Xenarios, I., Mendoza, L., and DeMicheli, G. 2008. Synchronous versus asynchronous modeling of gene regulatory networks. *Bioinformatics* 24, 1917–1925.

Gat-Viks, I., and Shamir, R. 2003. Chain functions and scoring functions in genetic networks. *Bioinformatics* 19, i109–i117.

Goles, E., and Salinas, L. 2010. Sequential operator for filtering cycles in Boolean networks. *Advances in Applied Mathematics* 45, 346–358.

Harris, S.E., Sawhill, B.K., Wuensche, A., and Kauffman, S. 2002. A model of transcriptional regulatory networks based on biases in the observed regulation rules. *Complexity* 7, 23–40.

Hirsch, E.A. 2000. New worst-case upper bounds for SAT. *Journal of Automated Reasoning* 24, 397–420.

Irons, D.J. 2006. Improving the efficiency of attractor cycle identification in Boolean networks. *Physica D* 217, 7–21.

Jarrah, A.S., Raposa, B., and Laubenbacher, R. 2007. Nested canalyzing, unate cascade, and polynomial functions. *Physica D* 233, 167–174.

Kauffman, S.A. 1993. *The Origins of Order: Self-organization and Selection in Evolution.* Oxford Univ. Press, New York.

Kauffman, S.A., Peterson, C., Samuelsson, B., and Troein, C. 2003. Random Boolean network models and the yeast transcriptional network. *Proc. Natl. Acad. Sci.* USA 100, 14796–14799.

Kauffman, S.A., Peterson, C., Samuelsson, B., and Troein, C. 2004. Genetic networks with canalyzing Boolean rules are always stable. *Proc. Natl. Acad. Sci. USA* 101, 17102–17107.

Leone, M., Pagnani, A., Parisi, G., and Zagordi, O. 2006. Finite size corrections to random Boolean networks. *Journal of Statistical Mechanics: Theory and Experiment* 2006, P12012.

Melkman, A., Tamura, T., and Akutsu, T. 2010. Determining a singleton attractor of an AND/OR Boolean network in $O(1.587^n)$ time. *Information Processing Letters* 110, 565–569.

Samuelsson, B., and Troein, C. 2003. Superpolynomial growth in the number of attractors in kauffman networks. *Physical Review Letters* 90, 098701.

Szallasi, Z., and Liang, S. 1998. Modeling the normal and neoplastic cell cycle with realistic Boolean genetic networks: Their application for understanding carcinogenesis and assessing therapeutic strategies, 66–76. *In Proc. Pacific Symposium on Biocomputing 1998*, World Scientific, Singapore.

Tamura, T., and Akutsu, T. 2009. Detecting a singleton attractor in a Boolean network utilizing SAT algorithms. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences* E92-A, 493–501.

Tamura, T., and Akutsu, T. 2009. Algorithms for singleton attractor detection in planar and nonplanar AND/OR Boolean networks. *Mathematics in Computer Science* 2, 401–420.

Tovey, C. A. 1984. A simplified NP-complete satisfiability problem. *Discrete Applied Mathematics* 8, 85–89.

Yamamoto, M. 2005. An improved $\tilde{O}(1.234^m)$-time deterministic algorithm for SAT, 644–653. *In Proc. 16th International Symposium on Algorithms and Computation*, Springer, New York.

Yamamoto, M. 2009. An $\tilde{O}(1.732^n)$ time algorithm for singleton attractor detection in AND/OR Boolean networks. *Manuscript*.

Zhang, S-Q., Hayashida, M., Akutsu, T., Ching, W-K., and Ng, M. K. 2007. Algorithms for finding small attractors in Boolean networks. *EURASIP Journal on Bioinformatics and Systems Biology* 2007, 20180.

APPENDIX

A1 Details for obtaining the running time of NC-SAT-Attractor(N)

For obtaining the running time of Algorithm **NC-SAT-Attractor**(N), we analyze an upper bound of $f(k,m)$ with respect to $k+m$. We estimate an upper bound of $f(k,m)$ with respect to k and m so that $f(k,m) = O^*(a^k \cdot b^m)$ for constants $a > 1$ and $b > 1$, where the two parameters a and b are any constants satisfying the recursions for $f(k,m)$. From this, the running time can be expressed as

$$\max_{0 \le \alpha \le 1} \left\{ \min\{2^{\alpha n}, a^{\alpha n} b^{(1-\alpha)n}\} \right\}.$$

The prospective of this expression is that a and b may depend on α. However, with more precise analysis of the above expression, we only have as much as the bound obtained by the simple analysis.

Recall that the function $f(k,m)$ is specified by the following recursions:

$$f(k,m) \;\le\; \max\left\{\begin{array}{l} f(k-1,m-1)+f(k-2,m)+\mathrm{poly}(k,m), \\ f(k-1,m-2)+f(k-1,m-1)+\mathrm{poly}(k,m) \end{array}\right\}.$$

If $f(k,m)$ is assumed to be exponential in both of k and m, we may assume that $f(k,m) = O^*(a^k \cdot b^m)$ for constants $a > 1$ and $b > 1$. (In what follows, we focus on exponential factors so that $f(k,m) \le a^k \cdot b^m$.) Suppose $f(k',m') \le a^{k'} \cdot b^{m'}$ for all k', m' such that $k'+m' < k+m$ with $k' \le k$ and $m' \le m$. From the recursions above,

$$f(k,m) \;\le\; \max\left\{\begin{array}{l} a^{k-1}b^{m-1}+a^{k-2}b^{m}, \\ a^{k-1}b^{m-2}+a^{k-1}b^{m-1} \end{array}\right\}.$$

Thus, we need any constants a and b satisfying

$$\left\{\begin{array}{lcl} a^{k-1}b^{m-1}+a^{k-2}b^{m} & \le & a^k b^m, \\ a^{k-1}b^{m-2}+a^{k-1}b^{m-1} & \le & a^k b^m, \end{array}\right.$$

which is equivalent to

$$\left\{\begin{array}{lcl} a+b & \le & a^2 b \\ 1+b & \le & ab^2 \end{array}\right.$$

From the first inequality, we have $a/(a^2-1) \le b$. From the second inequality, we have $b \le (1-\sqrt{4a+1})/(2a)$ and $(1+\sqrt{4a+1})/(2a) \le b$. Since $(1-\sqrt{4a+1})/(2a) < 0$ for $a > 0$, we have $(1+\sqrt{4a+1})/(2a) \le b$ from the second. Thus, we have

$$\left\{\begin{array}{lcl} b & \ge & a/(a^2-1) \\ b & \ge & (1+\sqrt{4a+1})/(2a) \end{array}\right.$$

Two functions $a/(a^2-1)$ and $(1+\sqrt{4a+1})/(2a)$ are decreasing for $a > 1$, and meet only at $a_0 \approx 1.4903$ for $a > 1$ where a_0 is the unique solution of $x^4 - 2x^2 - x + 1 = 0$ and $x > 1$. Note here that $a_0 > \sqrt{2}$. Furthermore, $a/(a^2-1) > 1$ for $1 < a \le a_0$, and $(1+\sqrt{4a+1})/(2a) > 1$ for $a_0 \le a < 2$. Thus, we may assume that $a > 1$ and $b > 1$ are *any* constants such that

$$(*)\ \cdots\ \left\{\begin{array}{lclcll} b & = & b_1 & \overset{\mathrm{def}}{=} & a/(a^2-1) & \text{for } 1 < a \le a_0 \\ b & = & b_2 & \overset{\mathrm{def}}{=} & (1+\sqrt{4a+1})/(2a) & \text{for } a_0 \le a < 2 \end{array}\right.$$

While $f(k,m)$ is one of the upper bounds of the running time for k variables and m constraints, we also have the obvious upper bound 2^k. Let $n = \alpha k$ and $m = (1-\alpha)n$. Thus, the running time of the algorithm is at most

$$\max_{0 \le \alpha \le 1} \{\min\{2^\alpha, (a/b)^\alpha b\}\}$$

to the power n, where a and b satisfy $(*)$. Note that in this expression, two parameters a and b may depend on α if these satisfy $(*)$. The bound $2^{n/2}$ in the proof of Theorem 1 is obtained by setting $a = \sqrt{2}$ and $b = \sqrt{2}/(\sqrt{2}^2 - 1) = \sqrt{2}$ (independent of α).

We will see that the least value of the expression above is $\sqrt{2}$, which is attained by setting, e.g., $a = b = \sqrt{2}$ as above. This fact is proved by showing that for a fixed value of $\alpha : 0 \leq \alpha \leq 1$,

$$\forall a, b > 1 \text{ s.t. } (*) \ \left[\min\{2^{\alpha}, (a/b)^{\alpha} b\} \geq \sqrt{2}\right].$$

For this, set $\alpha = 3/4$. Then, it suffices to show that for every $a > 1$ and $b > 1$ such that $(*)$ $(a/b)^{3/4} b \geq \sqrt{2}$. By elementary calculations, we have $(a/b_1)^{3/4} b_1 \geq \sqrt{2}$ for $1 < a \leq a_0$, and $(a/b_2)^{3/4} b_2 \geq \sqrt{2}$ for $a_0 \leq a < 2$.

The value of $\alpha = 3/4$ is obtained as follows: Consider the case of $1 < a \leq a_0$. Let $(a/b_1)^{\alpha} b_1 = \sqrt{2}$, that is, $(a^2 - 1)^{\alpha} (a/(a^2 - 1)) = \sqrt{2}$, which is equivalent to

$$\alpha = \frac{\ln \frac{\sqrt{2}(a^2-1)}{a}}{\ln(a^2 - 1)} \overset{\text{def}}{=} g(a).$$

Then, $\lim_{a \to \sqrt{2}} g(a) = 3/4$.

Figures

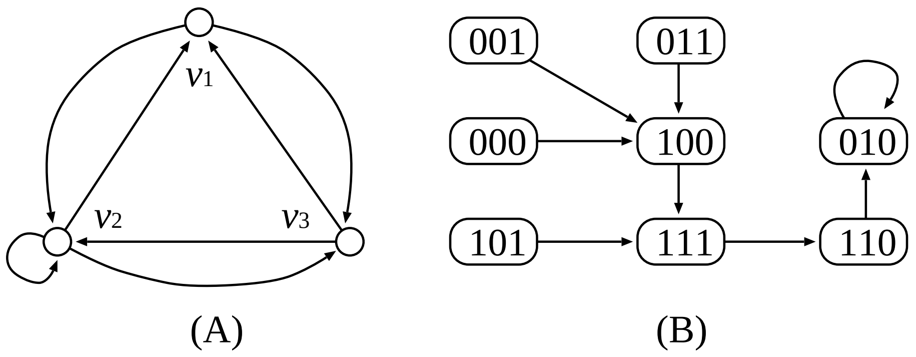

Figure 1: (A) Example of BN with nc-functions $v_1(t+1) = \overline{v_2(t)} \vee v_3(t)$, $v_2(t+1) = v_1(t) \vee (v_2(t) \wedge \overline{v_3(t)})$ and $v_3(t+1) = v_1(t) \wedge \overline{v_2(t)}$, (B) Diagram showing transition of the states of this BN, where $\langle 0, 1, 0\rangle$ is a singleton attractor.

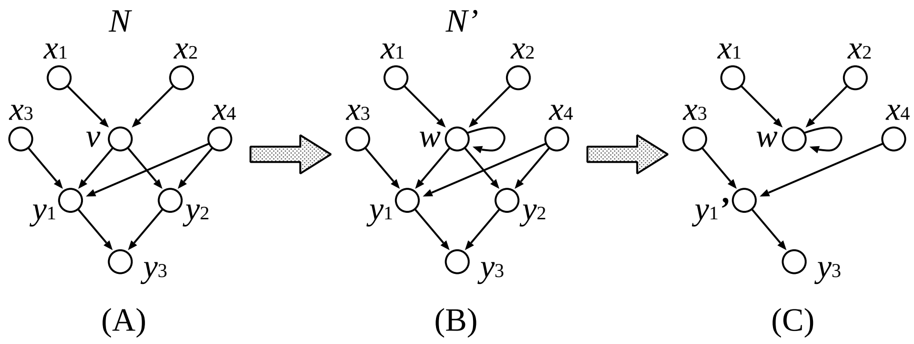

Figure 2: Example of **Convert2Constraint** ((A) $\Longrightarrow$ (B)) and **Reduce** ((B) $\Longrightarrow$ (C)).

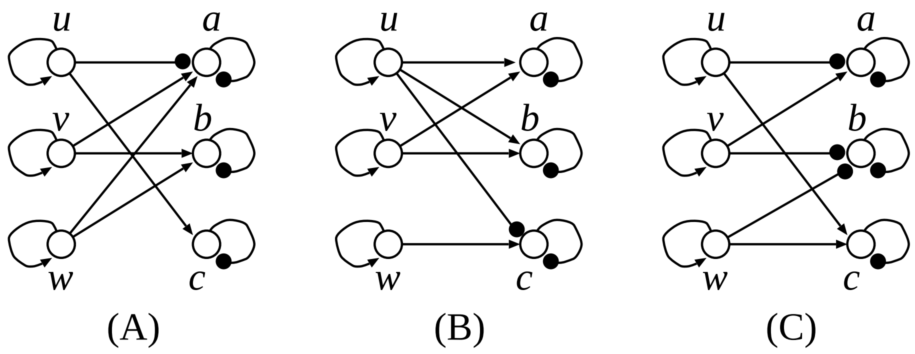

Figure 3: Example of 3 cases described in **NC-SAT-Attractor**(N). Usual arrow and arrow with '•' correspond to positive literal and negative literal, respectively. Only edges corresponding to literals in initial clauses are shown.