11-2008

Symbolic Approaches for Finding Control Strategies in Boolean Networks

Christopher J. Langmead
Carnegie Mellon University

Sumit Kumar Jha
Carnegie Mellon University

Journal of Bioinformatics and Computational Biology

Symbolic Approaches for Finding Control Strategies in Boolean Networks

Christopher James Langmead*
Carnegie Mellon University
5000 Forbes Ave.,
Pittsburgh, PA 15213, USA
E-mail: cjl@cs.cmu.edu

Sumit Kumar Jha
Carnegie Mellon University
5000 Forbes Ave.,
Pittsburgh, PA 15213, USA
E-mail: sumit.jha@cs.cmu.edu

We present an exact algorithm, based on techniques from the field of *Model Checking*, for finding control policies for Boolean networks (BN) with control nodes. Given a BN, a set of starting states, I, a set of goal states, F, and a target time, t, our algorithm automatically finds a sequence of control signals that deterministically drives the BN from I to F at, or before time t, or else guarantees that no such policy exists. Despite recent hardness-results for finding control policies for BNs, we show that, in practice, our algorithm runs in seconds to minutes on over 13,400 BNs of varying sizes and topologies, including a BN model of embryogenesis in *Drosophila melanogaster* with 15,360 Boolean variables. We then extend our method to automatically identify a set of Boolean transfer functions that reproduce the qualitative behavior of gene regulatory networks. Specifically, we automatically learn a BN model of *D. melanogaster* embryogenesis in 5.3 seconds, from a space containing 6.9×10^{10} possible models.

Keywords: Systems Biology; Synthetic Biology; Model Checking; Control; Boolean Networks

1. Introduction

Computational cellular and systems modeling is playing an increasingly important role in biology, bioengineering, and medicine. Boolean Networks (BN) are a popular coarse-grained abstraction of regulatory dynamics. In this paper, we consider the problem of automatically devising control policies for BNs. That is, given a BN model with external controls, we seek a sequence of control signals that will drive the network to a pre-specified state at (or before) a pre-specified time. This problem arises in fields such as Synthetic Biology and Medicine where the ability to control a system's behavior is essential.

*Corresponding author: cjl@cs.cmu.edu

Recently, it has been shown that finding control strategies for arbitrary BNs is NP-hard[1], but that polynomial-time algorithms exist for deterministic BNs if the network topology forms a tree. In this paper, we present an algorithm that can be applied to BNs with arbitrary network topologies. Our algorithm uses techniques from the field of *Model Checking*[13]. Model Checking refers to a family of algorithms and data structures for verifying systems of concurrent reactive processes. Historically, Model Checking has been used to verify the correctness and safety of circuit designs, communications protocols, device drivers, and C or Java code. Abstractions of these systems can be encoded as finite-state models that are equivalent to Boolean networks. We show that existing Model Checking algorithms can be used to find control strategies for BNs.

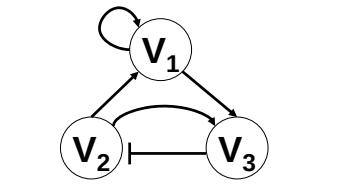
Two important features of Model Checking algorithms are that they are exact *and* scale to real-world problem instances. For example, Model Checking algorithms for finite-state systems have been able to reason about systems having more than 10^{20} states since 1990[7], and have been applied to systems with as many as 10^{120} states[6]. In this paper, we will show that Model Checking can be used to devise control strategies for very large Boolean networks (up to 15,360 nodes) within seconds or minutes. We will also show that our method can be used to automatically find BNs that reproduce the qualitative behavior of gene regulatory networks. The algorithm presented in this paper is relevant to a variety of fields, including Systems Biology, Synthetic Biology, and Medicine, and can potentially be used as a starting point for finding control policies in finer-grained abstractions of biological processes.

The format of this paper is as follows: We define Boolean networks and the control problem in Section 2. Next, we introduce concepts from the field of Model Checking in Section 3. We introduce our algorithm in Section 4 and related work in Section 5. We present and discuss the results of using our method in Section 6, and then summarize our method and discuss future work in Section 7.

2. Boolean Networks

A BN is a pair, $B = (\mathcal{G}, \Psi)$, where $\mathcal{G} = \{V, E\}$ is a directed graph, and $\Psi = \{\psi_1, \psi_2, ..., \psi_{|V|}\}$ is a set of Boolean transfer functions that collectively define the discrete dynamics of the network. Each vertex, $v_i \in V$, represents a Boolean random variable. The state of variable v_i at discrete time t is denoted by $v_i(t)$. The state of all vertices at time t is denoted by $\mathbf{v}(t)$. The directed edges in the graph specify causal relationships between variables. Let $Par(v_i) \subset V$ be the parents of v_i in the directed graph and let $k_i = |Par(v_i) \cup \{v_i\}|$. A node can be its own parent if we add a self-edge. Each Boolean *transfer function* $\psi_i : \{0,1\}^{k_i} \mapsto \{0,1\}$ defines the discrete dynamics of v_i from time t to $t+1$, based on the state of its parents at time t. Thus, the set Ψ defines the discrete dynamics of the entire BN. An example BN is shown in Figure 1-A. Notice that a BN is simply a compact encoding of a transition relation over V (Fig 1-B).

This basic model can be extended to define a BN with external controls by



$\psi_1 \equiv V_1(t+1) = V_1(t) \wedge V_2(t)$
$\psi_2 \equiv V_2(t+1) = \neg V_3(t)$
$\psi_3 \equiv V_3(t+1) = V_1(t) \wedge V_2(t)$

(A)

time t			time t + 1		
V_1	V_2	V_3	V_1	V_2	V_3
0	0	0	0	1	0
0	0	1	0	0	0
0	1	0	0	1	0
0	1	1	0	0	0
1	0	0	0	1	0
1	0	1	0	0	0
1	1	0	1	1	1
1	1	1	1	0	1

(B)

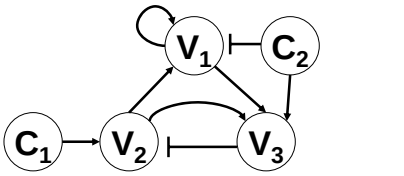
Fig. 1. (**A**) A Boolean Network (BN). A BN consists of a graph and a set of Boolean functions. The vertices of the graph correspond to Boolean variables and the edges describe functional dependencies. Arrowheads (resp. lines) represent promoting (resp. inhibiting) relationships. The Boolean functions describe the evolution of the model from time t to $t+1$. The functions can contain any combination of Boolean connectives. (**B**) A transition relation encoding the same dynamics as the BN. Notice that the BN is a compact encoding of the transition relation.

augmenting our graph with special control nodes, $\mathcal{G} = \{V, C, E\}$. Each control node, c_i, is connected to one or more nodes in V by a directed edge going from c_i to v_j (Fig. 2). Control nodes have no parents and represent externally manipulatable variables.

Consider a set of initial states, I, for the nodes in V specified in terms of a *characteristic function*. For example, the expression $I = (v_1 \wedge \neg v_2 \wedge v_3)$ defines the set $\{(1,0,1)\}$, and $I = (v_1 \wedge v_3)$ defines the set $\{(1,0,1),(1,1,1)\}$. We define a set of goal states, F, in a similar fashion. A *control policy*, $\Gamma = \langle \mathbf{c}(0), \mathbf{c}(1), ..., \mathbf{c}(t) \rangle$, is a set of Boolean vectors that defines a sequence of signals to be applied to the control nodes. The BN control problem is to find a control policy that drives the BN such that $\mathbf{v}(0) = I$ and $\mathbf{v}(t) = F$. Our goal in this paper is to algorithmically generate control policy, Γ, for a given Boolean network, B, initial set of state, I, a set of goal states, F, and end time, t, — or to guarantee that no such policy exists.

3. Model Checking

The term *Model Checking*[13] refers to a family of techniques from the formal methods community for verifying properties of complex systems, such as digital circuits. Given a model, $\mathcal{M}$, a set of starting states, S_0, and a proposition, ϕ, the *model checking problem* is to determine whether $\mathcal{M}, S_0 \models \phi$. Here, the symbol '$\models$' is read as "models" or "satisfies". In Model Checking, propositions are generally specified as formulae in temporal logic, which is a formalism for representing and reasoning about propositions qualified in terms of time. A complete discussion of Model Checking theory and practice is beyond the scope of this paper. The interested reader is directed to [13] for a detailed treatment of the subject.



$\psi_1 \equiv V_1(t+1) = V_1(t) \wedge V_2(t) \vee \neg C_2(t)$

$\psi_2 \equiv V_2(t+1) = \neg V_3(t) \wedge C_1(t)$

$\psi_3 \equiv V_3(t+1) = V_1(t) \wedge V_2(t) \wedge C_2(t)$

(A)

	V_1	V_2	V_3
Start (t=0)	0	0	0
Goal(t=3)	1	0	0

t	V_1	V_2	V_3	C_1	C_2
0	0	0	0	1	0
1	1	1	0	0	1
2	1	0	1	0	0
3	1	0	0		

(B)

Fig. 2. (**A**) A BN with two control nodes (C_1 and C_2). (**B-top**) An initial state and time-sensitive goal. (**B-bottom**) A control policy (last two columns) that achieves the goal at the specified time.

The field of Model Checking was born from a need to formally verify the correctness of hardware designs. Since its inception in 1981, it has expanded to encompass a wide range of techniques for formally verifying a variety of kinds of finite-state and hybrid (i.e., mixtures of finite and continuous variables) transition systems, including those with non-deterministic (i.e., asynchronous) and stochastic dynamics. Model Checking algorithms are simultaneously theoretically interesting and useful in practice. Significantly, they have become the preferred method for formal verification in industrial settings over traditional verification methods like theorem proving, which often need guidance from an expert human user.

Model Checking algorithms are exhaustive in the sense that they consider every possible behavior of the system. However, the power of Model Checking algorithms lies in the fact that most are based on an *implicit* search through the space of behaviors. These implicit methods, which are discussed further in Section 3.3, often allow Model Checking algorithms to verify properties many orders of magnitude larger than those that can be verified using an explicit search. We note that Model Checking algorithms are not a panacea; there are systems where even implicit methods do not work. In practice, however, Model Checking is often extremely successful, as evidenced by its industrial applications.

Another important capability of Model Checking algorithms is that they are capable of producing a *counterexample* to the specified property, if the property does not hold. We use this counterexample-generating capability in this paper in order to find control policies. Briefly, our properties express the notion that no control policy exists to achieve a particular goal. If the Model Checking algorithm finds a counter example, it provides the desired control policy. Alternatively, if the Model Checking algorithm cannot find a counter-example, we know that no such counter-example exists, due to the exhaustive nature of Model Checking.

Our approach to finding control policies for BNs is based on the observation that the problem of finding a control policy for BNs can be reduced to the Model Checking problem. Moreover, since we seek control policies which, by definition, seek to drive the system to a desired state within a finite amount of time, we can reduce our problem to that of performing *Bounded Model Checking*[5]. That is, we can utilize existing algorithmic approaches for solving the Bounded Model Checking problem to automatically identify control policies, or to formally prove that no policy exists. In the remainder of this section, we will outline the key elements of Model Checking, which include: i) a formal specification of the transition system (Sec. 3.1); ii) a formal specification of temporal properties in temporal logic (Sec. 3.2); and iii) algorithms for formally verifying that a given model satisfies a given property (Sec. 3.4). Additionally, we will introduce concepts relevant to *Symbolic Model Checking* in Section 3.3.

3.1. *Modeling Concurrent Systems as Kripke Structures*

An *atomic proposition*, a, is a Boolean predicate referring to some property of a given system. Let AP be a set of atomic propositions. A *Kripke structure*, $\mathcal{M}$, over AP is a tuple, $\mathcal{M} = (S, R, L)$. Here, S is a finite set of states, $R \subseteq S \times S$ is a total transition relation between states, and $L : S \mapsto 2^{AP}$ is a labeling function that labels each state with the set of atomic propositions that are true in that state. Variations on the basic Kripke structure exist. For example, if the system is stochastic, then we replace the transition relation, R, with a stochastic transition matrix, T where element $T(i, j)$ contains either a transition rates (for continuous-time Markov models) or a transition probability (for discrete-time Markov models).

It is easy to see that, in principle, BNs can be encoded as Kripke structures. The state space, S, corresponds to the $2^{|V \cup C|}$ possible states of the BN . We will use the atomic propositions to reveal the state of each variable in the model. That is, $|AP| = |V \cup C|$ and the propositions will be of the form: "*is the state of* v_i 1?" The labeling function, L, can thus be used to define the set of initial states, I, and goal states, F (see Sec. 2). The transition relation, R, corresponds to the table in Figure 1-B. Naturally, it is generally not possible to explicitly instantiate the Kripke structure for an arbitrary BN because the state space is exponential in the number of nodes. We will discuss how Kripke structures can be efficiently encoded symbolically in Section 3.3.

3.2. *Temporal Logics*

Temporal logic is a formalism for describing behaviors in finite-state systems. It has been used since 1977 to reason about the properties of concurrent programs[23]. There are a number of different temporal logics from which to chose, and different logics have different expressive powers. In this paper, we use a small fragment of the Computation Tree Logic (CTL). CTL formulae can express properties of *computation trees*. The root of a computation tree corresponds to the set of initial

states (i.e., I) and the rest of the (infinite) tree corresponds to all possible paths from the root. A complete discussion of CTL and temporal logics is beyond the scope of this paper. The interested reader is directed to [13] for more information.

The syntax of CTL is given by the following minimal grammar:

$$\phi ::= a \mid true \mid (\neg\phi) \mid (\phi_1 \wedge \phi_2) \mid \mathbf{EX}\phi \mid \mathbf{E}[\phi_1 \mathbf{U} \phi_2] \tag{1}$$

Here, $a \in AP$, is an atomic proposition; "true" is a Boolean constant; $\neg$ and $\vee$ are the normal logical operators; $\mathbf{E}$ is the existential path quantifier (i.e., "there exists some path from the root in the computation tree"); and $\mathbf{X}$ and $\mathbf{U}$ are temporal operators corresponding to the notions of "in the next state", and "until", respectively. Given these, additional operators can be derived. For example, "false" can be derived from "$\neg true$" and the universal quantifier, $\mathbf{AX}\phi$, can be defined as $\neg\mathbf{EX}\neg\phi$.

Given some path through the computation tree, $\pi = \langle \pi[0], \pi[1], \ldots \rangle$, the semantics of a CTL formula are defined recursively:

$\pi \models a$ iff a $\in$ L($\pi[0]$)
$\pi \models true, \forall\pi$
$\pi \models \neg\phi$ iff $\pi \not\models \phi$
$\pi \models \phi_1 \wedge \phi_2$ iff $\pi \models \phi_1$ and $\pi \models \phi_2$
$\pi \models \mathbf{EX}\phi$ iff $\pi[1] \models \phi$
$\pi \models \mathbf{E}[\phi_1 \mathbf{U} \phi_2]$ iff $\exists i \geq 0, \pi[i] \models \phi_2 \wedge \forall j < i, \pi[j] \models \phi_1$

3.3. *Symbolic Encodings of Kripke Structures*

The phrase *Symbolic Model Checking* refers to any technique whereby sets of states and the transitions between state are represented *implicitly* using Boolean functions. As a trivial example, consider a toy system with two binary variables, v_1 and v_2. If the set of allowable states for this system is $\{(11), (01), (10)\}$, we can efficiently encode this set of states using the following *characteristic function*: $v_1 \vee v_2$. Such Boolean formulas can be efficiently represented and manipulated in two ways: using Binary Decision Diagrams (BDDs), or using propositional satisfiability (SAT) formulae. In practice, Symbolic Model Checking algorithms based on SAT encodings generally scale to larger systems than those based on BDD encodings. This is primarily due to the efficiency and power of modern-day propositional SAT solvers[a]. Of course, SAT is *the* prototypical NP-hard problem, which implies that one cannot guarantee that SAT-based Model Checking algorithms will always succeed. However, empirically, we have found that SAT-based methods scale to very large BNs, as will be shown in Section 6.

We note that, in practice, the construction of symbolic encodings of Kripke structures is done automatically from a high-level language describing the finite-

[a]It is worth noting that modern SAT solvers are capable of solving instances of SAT with hundreds of thousands of variables (or more), and millions of clauses[11].

state system and its behavior. That is, it is generally not necessary to first construct the explicit state space. This is important, because the systems we will consider are too large to represent explicitly. In this paper, we use the specification language used in the Symbolic Model Checking tool NuSMV[10].

SAT-based Encodings for Bounded Model Checking Bounded Model Checking algorithms perform a kind of iterative deepening depth-first search when searching for a counterexample to the property. The Kripke structure is "unrolled" for a fixed number of steps, k, and the algorithm terminates if the property is violated within k steps. This procedures starts with $k = 1$ and iteratively increases k until the property is verified, or a counterexample is found.

In the case of Bounded Model Checking, Kripke structures can be encoded symbolically as an instance of propositional satisfiability. SAT-solvers are then used to perform Model Checking. Given a model $\mathcal{M}$, a CTL formula, ϕ, and a bound k, it is possible to construct a propositional formula $\langle \mathcal{M}, \phi \rangle_k$ that will be satisfiable if and only if the property $\mathcal{M} \models^k \phi$. That is, if $\mathcal{M}$ satisfies ϕ for the first k steps (see [5] for a more detailed discussion).

The propositional formula, $\langle \mathcal{M} \rangle_k$, for an unrolled transition for a model $\mathcal{M}$ is defined as follows:

$$\langle \mathcal{M} \rangle_k := I(S_0) \wedge \bigwedge_{i=0}^{k-1} T(s_i, s_{i+1})$$

where $I(S_0)$ is the characteristic function of the set of initial states, and $T(s_i, s_{i+1})$ is the characteristic function of the transition relation.

It is also possible to form a propositional formula $\langle \phi \rangle_k$ that is true if and only if the formula ϕ is valid along a path of length k. The conjunction of $\langle \mathcal{M} \rangle_k$ and $\langle \phi \rangle_k$ gives us the propositional formula we seek that will be true if and only if $\mathcal{M} \models^k \phi$.

Example: Consider the BN in Fig. 1. Suppose we want to start from the state $(v_1 = 0, v_2 = 0, v_3 = 0)$ at time $t = 0$ and determine whether it can reach the state $(v_1 = 0, v_2 = 1, v_3 = 1)$ at time $t = 2$. The characteristic function for the initial state is thus $I(S_0) = (\neg v_1^0 \wedge \neg v_2^0 \wedge \neg v_3^0)$. The characteristic function describing the first step is: $T(s_0, s_1) = ((v_1^1 \leftrightarrow (v_1^0 \wedge v_2^0)) \wedge (v_2^1 \leftrightarrow \neg v_3^0) \wedge (v_3^1 \leftrightarrow (v_1^0 \wedge v_2^0)))$, where $\leftrightarrow$ represents the logical XNOR. Similarly, the characteristic function describing the second step is: $T(s_1, s_2) = ((v_1^2 \leftrightarrow (v_1^1 \wedge v_2^1)) \wedge (v_2^2 \leftrightarrow \neg v_3^1) \wedge (v_3^2 \leftrightarrow (v_1^1 \wedge v_2^1)))$. The characteristic function describing the property is simply $\phi = v_1^2 \wedge v_2^2 \wedge v_3^2$. Taking the conjunction of these characteristics we obtain:

$$\begin{aligned}\langle \mathcal{M}, \phi \rangle_k := (\neg v_1^0 \wedge \neg v_2^0 \wedge \neg v_3^0) \wedge ((v_1^1 \leftrightarrow (v_1^0 \wedge v_2^0)) \wedge (v_2^1 \leftrightarrow \neg v_3^0) \wedge (v_3^1 \leftrightarrow (v_1^0 \wedge v_2^0))) \\ \wedge ((v_1^2 \leftrightarrow (v_1^1 \wedge v_2^1)) \wedge (v_2^2 \leftrightarrow \neg v_3^1) \wedge (v_3^2 \leftrightarrow (v_1^1 \wedge v_2^1))) \wedge v_1^2 \wedge v_2^2 \wedge v_3^2.\end{aligned}$$

This formula cannot be satisfied, which is correct, since the property does not hold for the BN in Fig. 1. We could similarly derive a formula that incorporates the control nodes in Fig. 2. A satisfying assignment of such a formula, if it exists, would reveal a valid control policy.

```
MODULE BN
VAR
  V1: boolean;          // variable node 1
  V2: boolean;          // variable node 2
  V3: boolean;          // variable node 3
  C1: boolean;          // control node 1
  C2: boolean;          // control node 2
  COUNTER: 0 .. T+1;// counter
ASSIGN
  init(V1) := 1;
  init(V3) := 1;
  next(V1) := (V1 & V2) | !C2 ;
  next(V2) := ! V3 & C1 ;
  next(V3) := V1 & V2 & C2 ;
  next(COUNTER) := COUNTER+1 ;
```

Fig. 3. Pseudocode based on the language used in the Symbolic Model Checking program NuSMV. This code implements the BN in Figure 2. The code consists of a module with variable declaration statements, "init" statements that initialize the variables, and "next" statements that implement each ϕ_i and increment a counter.

3.4. *Model Checking Algorithms*

A Model Checking algorithm takes a Kripke structure, $\mathcal{M} = (S, R, L)$, and a temporal logic formula, ϕ, and finds the set of states in S that satisfy ϕ: $\{s \in S \mid \mathcal{M}, s \models \phi\}$. The complexity of Model Checking algorithms varies with the temporal logic and the operators used. For the types of formulas used in this paper (see Sec. 4), an explicit state Model Checking algorithm requires time $O(|\phi|(|S|+|R|))$, where $|\phi|$ is the number of sub-formulas in ϕ[13]. Of course, for very large state spaces, even linear time is unacceptable. Symbolic Model Checking algorithms operate either BDD or SAT encodings of the Kripke structure and CTL formula. SAT-based Bounded Model Checking algorithms rely on SAT solvers to find a satisfying assignment for the propositional formula, if one exists.

4. A Symbolic Model Checking Approach to Finding Control Policies

The use of Model Checking algorithms for finding control strategies requires three steps: *First*, the BN must be encoded using a high level language for describing finite-state models. Different Model Checking software use different modeling languages. In Figure 3, we show pseudo-code for encoding the BN in Figure 2. This pseudo-code is based on the language used in the model-checking tool NuSMV. The code contains a block of variable definitions. In the example, we declare Boolean variables for v_1, v_2, v_3, c_1,and c_2. The set of initial states, I, is encoded using "init" statements. The update rules, Ψ, are encoded using "next" statements. A single variable COUNTER is declared that marks the passage of time. A "next" statement for COUNTER updates the counter.

Second, a CTL formula must be written. In this paper, we are concerned with CTL formulae that ask whether it is possible to end up in the goal state(s), F, at time t. Let ϕ_F be a formula describing the goal state. This formula can describe any subset of the variables in the BN. For example, $\phi_F := v_1 \wedge \neg v_2 \wedge v_3$ or $\phi_F := v_1 \wedge v_3$

are both valid formulas. The former chooses to specify the state of each variable, the latter does not. Let ϕ_t :=COUNTER= t be a Boolean formula that evaluates to true if the variable COUNTER is t. The formula $\phi := \mathbf{E}[\neg\phi_F \mathbf{U}(\phi_F \wedge \phi_t)]$ can be used to find a control policy. In English, this formula says: "*There exists a path that enters state F for the first time at time t*". Alternatively, if we wish to relax the restriction that the BN cannot enter state F before time t, we would use the formula $\phi' := \mathbf{E}[true\mathbf{U}(\phi_F \wedge \psi_t)]$, which translates as "*In the future, the model will be in F at time t.*" Temporal logics are very expressive and can encode a number of complex behaviors. For example, it is possible to specify particular milestones through which the model should pass en route to the final goal. That is, one can construct formula that say that the BN should enter state X_1 before X_2, must enter X_2 by time t_1, and must reach the goal state at exactly time t_2. This expressive power is one of the key advantages of a Model Checking based approach to the design of control policies.

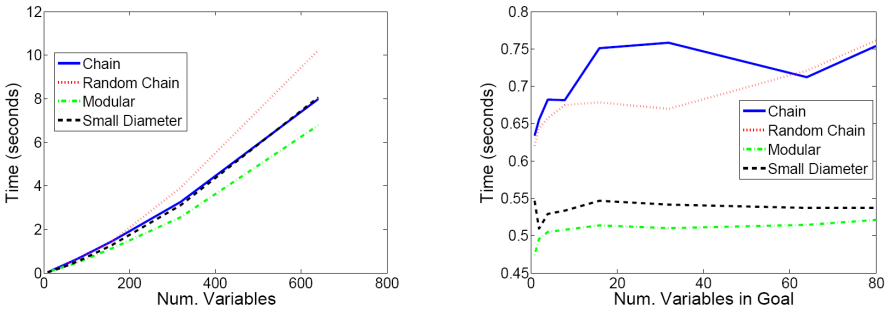
Finally, we apply an appropriate Symbolic Model Checking algorithm to find a control policy. If a control policy exists (i.e., if ϕ is true), then we ask the Model Checking algorithm for a *witness*, π_w, to the formula. The control policy, Γ, is then simply extracted from π_w by reading off the values of $\langle \mathbf{c}(0), \mathbf{c}(1), ..., \mathbf{c}(t) \rangle$[b].

5. Related Work

Boolean Networks have been used extensively to model complex biological systems (e.g., [2, 3, 16, 17]). The design of control strategies for Boolean networks and related models has been considered by a number of different authors (e.g.,[1, 9, 14]). Akutsu and co-workers[1] were the first to show that the design of control policies is NP-hard. They also provide a polynomial-time algorithm that works on the special case where the topology of the BN forms a tree. The primary difference between our work and these is that our method is based on Symbolic Model Checking and we place no restriction on the topology of the network. We will show in the next section that despite the fact that the problem is NP-hard, in practice Model Checking based approaches to control policy design can scale to very large models. We believe that this is the first application of Model Checking for this problem. Of course, the hardness result implies that our approach will not apply to every BN.

Recently, there has been growing interest in the application of formal methods, including Model Checking to biology. Most applications of Model Checking in biology have been directed to the verification of the properties of biochemical and regulatory networks (e.g.,[4, 8, 12, 18]), although not for the design of control policies. In our own work, we have applied Model Checking[20], and a related technology based on decision procedures[21] to the protein folding problem.

[b]Equivalently, as we performed in our experiments, we can request a counterexample to $\neg\phi$.



(A) (B)

Fig. 4. Average runtimes as a function of the size of the BN, (A), and the number of controls in the goal state, (B). Vertical axis represents runtime (in seconds). Horizontal axis refers to the varied parameter. In each plot, the four lines correspond to the four kinds of randomly generated BNs (see text,Sec. 6.1).

6. Results

We present results from three kinds of experiment. The first experiment is designed to highlight the scalability of a Model Checking based approach to control policy design. The second experiment applies our approach to an existing BN model of embryo development in *Drosophila*. The final experiment reduces the problem of learning Boolean transfer functions in BNs to a control problem, and then solves that problem using our method.

6.1. *Scalability*

We have performed a large-scale study on randomly generated BNs in order to characterize the scalability of our approach. In total, we considered 13,400 separate BNs. We considered several different network topologies intended to reflect different kinds of networks ranging from simple feedback loops (*chains*), feedback loops with complex topologies (*random chains*), loosely coupled modules (*modular*), to a dense network (*small diameter*). Chain are defined as those where the non-control nodes are isomorphic to a circular linked list. Random chains are chain graphs with randomly added long-range edges. Modular graphs consist of a collection of sparsely connected random subgraphs. Small diameter graphs consist of a collection of densely connected random subgraphs. Images of some representative topologies can be found at the following URL:`http://www.cs.cmu.edu/~cjl/publications.html`. Within each network category, we performed separate experiments randomly generating graphs by varying: a) the number of non-control variables over the interval [10,640]; b) the average number of parents for each node over the interval [2, 8]; c) the number of control nodes over the interval [2,64]; d) the number of variables specified in the goal state, F, over the interval [4,80]; and e) the target time, t, over the interval [1,32]. For each combination of parameters, we generated 100 BNs randomly, constructed a CTL formula, and identified a control strategy using NuSMV.

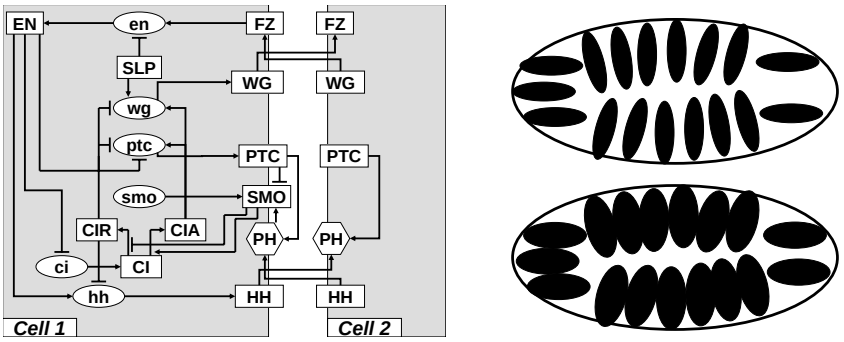


Fig. 5. **(Left)** The *Drosophila* segment polarity BN from Albert and Othmer. The figure shows one cell in detail (large grey box), and the inter-cellular signals (*WG* and *HH*) between two adjacent cells. See text for more details. **(Right)** Cartoon depiction of expression pattern of *wg* in wild-type (top) and a "broad-stripe" mutant embryo (bottom).

Due to space limitations, we simply report that each experiment took less than 12 minutes on a single Pentium 4 processor with 2 GB of memory. The mean and median runtimes were 2 and 0.6 seconds, respectively. The longest runtime (693 seconds) was on a random chain topology model with 80 nodes, an average in-degree of 4, 4 control nodes, a target specifying the state of 4 variables, and a time of 32 steps. It is not surprising that the longest runtime does not correspond to the largest network, since the complexity of a random instance of SAT is not uniquely determined by the number of variables. Figure 4 shows average runtimes as a function of the size of the network and the number of variables specified in the goal state. A more comprehensive set of figures can be found at the following URL:`http://www.cs.cmu.edu/~cjl/publications.html`. Our results suggests that a Model Checking approach to policy design scales well to randomly generated BNs.

6.2. *Application To* D. melanogaster *Embryo Development*

To test our approach on a BN for a real biological process, we applied it to the task of finding control policies to an existing model of fruit fly embryo development[3]. Briefly, Albert and Othmer have developed a BN model of the segment polarity gene network in *D. melanogaster* (Fig. 5-left). The model comprises 5 RNAs: (*wingless (wg)*; *engrailed (en)*; *hedgehog (hh)*; *patched (ptc)*; and *cubitus interruptus (ci)*), and 10 proteins: (*WG*; *EN*; *HH*; *PTC*; *CI*; *smoothened (SMO)*; *sloppy-paired (SLP)*; a transcriptional repressor, *(CIR)*, for *wg*, *ptc*, and *hh*; a transcriptional activator, *(CIA)* for *wg* and *ptc*; and the *PTC-HH* complex, (*PH*)). Each molecule is modeled as a Boolean variable and the update rules are Boolean formulas that take into account both intra-cellular state, and inter-cellular communication. The Albert and Othmer research did not consider the question of control policy design.

Albert and Othmer have demonstrated that the Boolean model accurately re-

produces both wild-type and mutant behaviors. In their experiments, they consider a 1-dimensional array of cells initialized to the experimentally characterized cellular blastoderm phase of *Drosophila* development, which immediately precedes the activation of the segment-polarity network. The purpose of the segment-polarity network is to maintain a pattern of expression throughout the life of the fly that defines the boundaries between *parasegments*, small linear groupings of adjacent cells. Two possible parasegment boundary expression patterns are shown in Figure 5-(right)[c]. In the Albert and Othmer work, the parasegments are four cells wide. We note that the steady-state expression patterns of different sub-populations of cells differ due to inter-cellular communication — this is precisely the mechanism by which the parasegment boundaries are maintained. That is, the fate of every cell is *not* the same, even though each cell is running the same regulatory network.

In our experiment, we modified the Albert and Othmer BN in two ways. *First*, we considered a 32x32, two-dimensional array of cells of dimension, instead of the 1x12 one-dimensional array of cells considered in [3]. We believe that this extension to a two-dimensional model is the first of its kind; we also believe that the 15,360 Boolean variables in our model is the largest ever considered for the purpose of control policy design. Adjacent cells in the network can communicate, which introduces loops in overall topology of the BN for the 32x32 array of cells. *Second*, we modified the network such that the RNAs *wg* and *hh* becomes a control node in the network. In principle, one could control RNAs through RNA-silencing or micro RNAs. We used our methods to design two control policies for *hh*. The first is designed to drive the system to either the wild-type expression pattern (Fig. 5-right (top)) and the other to a "broad-stripe" pattern (Fig. 5-right (bottom)). Example NuSMV input files for a 4×4 and a 32×32 array can be found at the following URL:`http://www.cs.cmu.edu/~cjl/publications.html`. Our algorithms successfully found the two control policies in 6.1 and 6.2 minutes, respectively. We believe these results strongly suggest that our approach can be used to find control signals for biologically relevant BNs of substantial size. We note that the Model Checking algorithm doesn't "know" that the model represents a 32x32 network of cells running the same logic. That is, it is not simply learning a control signal for a subset of the nodes and then applying it to the entire network.

6.3. *Learning Gene Regulatory Networks*

One of the most important challenges in the field of Systems Biology is the creation of dynamic models of biological systems from experimental observations. When quantitative measurements about reaction rates are scarce, qualitative models, like BNs, are often a suitable alternative. Learning BNs from data, however, is computationally challenging[22] due to the number of possible BNs over a given set of variables.

[c]Actual images of such expression patterns can be found at `http://www.fruitfly.org` and [25]

We considered a somewhat easier problem of identifying a set of transfer functions, one for each node in the BN (see Fig. 1), that reproduces a known behavior of the system. Here we assume that we know the topology of the BN (i.e., we know the parents of each node in the BN), but that we do *not* know the Boolean function that drives each node. That is, we want to learn the set Ψ. Computationally, this is challenging because there are an exponential number of possible Boolean functions for each node. For example, if the in-degree of a node n_i is k, there are 2^k possible joint-states of the parents of that node. We can thus encode the Boolean function ψ_i describing the dynamics of of node n_i using a truth table with 2^k rows. There are, however, $c = 2^{2^k}$ possible truth tables with 2^k rows. In a gene regulatory network, k is likely to be small, as it is in the fly network, where k is no larger than 4. Thus, c can be treated like a constant. Nevertheless, if there are c possible Boolean functions for each of n nodes, the number of Boolean networks is then $O(c^n)$.

Identifying a set of transfer functions can be reduced to a control policy design problem as follows: Given the network topology, $\mathcal{G}$, a starting state, I, and a known fix-point, F, we will create a set of $|V|$ control nodes, one for each node graph. The control nodes have c possible states, and their purpose is to select which of the c possible Boolean functions will drive the dynamics of node n. The value of each control node is set at the beginning of the execution of the model and held constant. Thus, there are $O(c^n)$ possible BNs. The control problem is to find a setting of the control nodes that drives the system from I to F within a specified interval of time. A solution to this problem will reveal a set of transfer functions that reproduce the desired behavior.

In the case of the fly network, $\mathcal{G}$, I, and F are all known. We used our method to learn the transfer functions as outlined in the previous paragraph using a target time of 10 steps. For this experiment, we used a 1x4 grid of cells. The resulting space of possible BNs had 6.9×10^{10} possible models. Our algorithm found a solution in 5.3 seconds using Bounded Symbolic Model Checking. These results suggest that Model Checking may relevant to automatically learning BNs. An obvious question is whether the same method could also be used to simultaneously learn the network topology and the transfer functions. This is an interesting question for further research.

7. Conclusions and Future Work

We have introduced a practical method for automatically discovering control sequences for Boolean networks based on techniques from the field of Model Checking. Our approach scales to very large BNs, having as many as 15,360 nodes, and runs in seconds to minutes. We have also shown that our method can be used to learn BNs. We note that, due to the inherent computational complexity of finding control policies in BNs[1], we cannot claim that our approach will scale to *every* BN of large size. One might ask whether the some basic property of biological networks might lead to easy instances. This is an interesting question and we believe it may be

related to the phenomenon of canalizing functions (also known as forcing functions) and other apparently generic properties of BNs of biological processes (e.g., [24]). Our intuition is that the presence of one or more canalizing functions in the BN might lead to easy instance of SAT.

BNs have been used widely to model a range of biological phenomena. However, the fact that BNs make strong assumptions about the binary nature of each variable (i.e., active or inactive), the synchronous nature of the updates, the assumption that time unfolds in discrete steps, and the assumption that the dynamic are deterministic. Ultimately, these assumptions limit the overall applicability of BNs in biology. We note, however, that our counterexample-based approach to control policy design and learning can be adapted for use to a much broader range of models including those with continuous-valued variables, asynchronous updates between variables, continuous time, and stochastic transitions[19]. We are presently pursuing these goals as part of ongoing research. Additional goals include the use of all-solutions SAT solver [e.g., 15] to identify every valid control policy. SAT-based techniques for identifying such things as periodic orbits have also been studied[26].

Acknowledgments

This research was supported by a U.S. Department of Energy Career Award (DE-FG02-05ER25696), and a Pittsburgh Life-Sciences Greenhouse Young Pioneer Award to C.J.L. The authors would like to thank Ed Clarke for his helpful comments and guidance.

References

1. T. Akutsu, M. Hayashida, W-K. Ching, and M.K. Ng. Control of boolean networks: Hardness results and algorithms for tree structured networks. *J. Theor. Biol.*, 244:670–679, 2007.
2. T. Akutsu, S. Miyano, and S. Kuhara. Inferring qualitative relations in genetic networks and metabolic pathways. *Bioinformatics*, 16(8):727–734, 2000.
3. R. Albert and H. G. Othmer. The topology of the regulatory interactions predics the expression pattern of the segment polarity genes in drosophila melanogaster. *J. Theor. Biol.*, 223:1, 2003.
4. M. Antoniotti, A. Policriti, N. Ugel, and B. Mishra. Model building and model checking for biochemical processes. *Cell Biochem Biophys.*, 38(3):271–286, 2003.
5. A. Biere, A. Cimatti, E. M. Clarke, M. Fujita, and Y. Zhu. Symbolic model checking using SAT procedures instead of BDDs. *Proc 36th ACM/IEEE Conf. on Design automation (DAC)*, pages 317–320, 1999.
6. J.R. Burch, E. M. Clarke, D. E. Long, K. L. McMillan, and D. L. Dill. Symbolic model checking for sequential circuit verification. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 3(4):401–424, 1993.
7. J.R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Proc. Fifth Ann. IEEE Symposium on Logic in Computer Science*, pages 428–439, 1990.
8. N. Chabrier and F. Fages. Symbolic Model Checking of Biochemical Networks. *Proc 1st Internl Workshop on Computational Methods in Systems Biology*, pages 149–162, 2003.

9. P.C. Chen and J.W. Chen. A markovian approach to the control of genetic regulatory networks. *Biosystems*, 90(2):535–45, 2007.
10. A. Cimatti, E.M. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. Nusmv 2: An opensource tool for symbolic model checking. In *Proc. 14th International Conference on Computer Aided Verification (CAV02)*, pages 359–364, 2002.
11. E. M. Clarke, A. Fehnker, Z. Han, B. Krogh, J. Ouaknine, O. Stursberg, and M. Theobald. Abstraction and counterexample-guided refinement in model checking of hybrid systems. *Int. J. of Found. of Comp. Sci.*, 14(4):583–604, 2003.
12. E.M. Clarke, J.R. Faeder, L.A. Harris, C.J. Langmead, A. Legay, and S.K. Jha. Statistical model checking in biolab: Application to the automated analysis of the t-cell receptor signalling pathway. *Proc. 6th Conf. on Computational Methods in Systems Biology, (CMSB)*, pages 231–250, 2008.
13. E.M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, Cambridge, MA, 1999.
14. A. Datta, A. Choudhary, M. L. Bittner, and E.R. Dougherty. External control in markovian genetic regulatory networks. *Mach. Learn.*, 52(1-2):169–191, 2003.
15. O. Grumberg, A. Schuster, and A. Yadgar. Memory efficient all-solutions sat solver and its application for reachability analysis. In *Proc. 5th Int. Conf. on Formal Methods in Computer-Aided Design (FMCAD)*, pages 275–289, 2004.
16. S.E. Harris, B.K. Sawhill, A. Wuensche, and S. Kauffman. A model of transcriptional regulatory networks based on biases in the observed regulation rules. *Complex.*, 7(4):23–40, 2002.
17. S. A. Kauffman. *The Origins of Order: Self-Organization and Selection in Evolution*. Oxford University Press, 1993.
18. M. Kwiatkowska, G. Norman, D. Parker, O. Tymchyshyn, J. Heath, and E. Gaffney. Simulation and verification for computational modelling of signalling pathways. *Proc. 38th Conf. on Winter simulation (WSC)*, pages 1666–1674, 2006.
19. C. J. Langmead. Towards inference and learning in dynamic bayesian networks using generalized evidence. Technical Report CMU-CS-08-151, Carnegie Mellon University, Department of Computer Science, 2008.
20. C.J. Langmead and S. K. Jha. Predicting protein folding kinetics via model checking. In *Lecture Notes in Bioinformatics: The 7th Workshop on Algorithms in Bioinformatics (WABI)*, pages 252–264, 2007.
21. C.J. Langmead and S.K. Jha. Using Bit Vector Decision Procedures for Analysis of Protein Folding Pathways. In *Proc. 4th Workshop on Constraints in Formal Verification (CFV)*, 2007.
22. D. Nam, S. Seo, and S. Kim. An efficient top-down search algorithm for learning boolean networks of gene expression. *Mach. Learn.*, 65(1):229–245, 2006.
23. A. Pnueli. The temporal logic of programs. In *Proc. 18th IEEE. Foundations of Computer Science (FOCS)*, pages 46–57, 1977.
24. I. Shmulevich, H. Lhdesmki, E. R. Dougherty, J. Astola, and W. Zhang. The role of certain post classes in boolean network models of genetic networks. *Proc Natl Acad Sci U S A*, 100(19):10734–10739, 2003.
25. T. Tabata, S. Eaton, and T. B. Kornberg. The drosophila hedgehog gene is expressed specifically in posterior compartment cells and is a target of engrailed regulation. *Genes Dev.*, 6(12B):2635–2645, 1992.
26. I. Zinovik, D. Kroening, and Y. Chebiryak. An algebraic algorithm for the identification of Glass networks with periodic orbits along cyclic attractors. In *Proc 2nd Conf on Algebraic Biology (AB07)*, pages 140–154, 2007.