Abstract—Probabilistic Boolean networks (PBNs) have previously been proposed so as to gain insights into complex dynamical systems. However, identification of large networks and their underlying discrete Markov chain which describes their temporal evolution still remains a challenge. In this paper, we introduce an equivalent representation for PBNs, the stochastic conjunctive normal form network (SCNFn), which enables a scalable learning algorithm and helps predict long-run dynamic behavior of large-scale systems. State-of-the-art methods turn out to be 400 times slower for middle-sized networks (i.e., containing 100 nodes) and incapable of terminating for large networks (i.e., containing 1000 nodes) compared to the SCNFn-based learning, when attempting to achieve comparable accuracy. In addition, in contrast to the currently used methods which introduce strict restrictions on the structure of the learned PBNs, the hypothesis space of our training paradigm is the set of all possible PBNs. Moreover, SCNFn enable efficient sampling so as to statistically infer multistep transition probabilities which can provide information on the activity levels of individual nodes in the long run. Extensive experimental results showcase the scalability of the proposed approach both in terms of sample and runtime complexity. In addition, we provide examples to study large and complex cell signaling networks to show the potential of our model. Finally, we suggest several directions for future research on model variations, theoretical analysis, and potential applications of SCNFn.

Index Terms—Dynamical system, learning systems, Markov chain, network identification, probabilistic Boolean network (PBN), statistical inference.

I. INTRODUCTION

Many complex systems that have recently received intense research attention by the scientific community fall in the broad category of dynamical systems. A dynamical system is typically governed by rules that describe the time dependence of a set of variables. Nonlinear (or Linear) dynamical systems which are, in general, described by a system of nonlinear (or respectively, linear) differential equations can often be reconstituted and analyzed using a qualitative or semiquantitative estimate of the behavior of their state variables. In situations where the exact values of the states are not required, estimating the behavior of such systems using binary logical models can be extremely fast when compared to learning and simulating complicated systems of differential equations using numerical methods.

Boolean networks (BNs), originally proposed by Kauffman [1], [2], constitute a very-well-studied qualitative modeling framework, which has been used for multifarious applications such as genetic regulatory networks [1], [3], neural networks [4], [5], robotics [6], and econometrics [7]. The probabilistic BN (PBN) paradigm was introduced by Shmulevich et al. [8] as a semiquantitative extension of BN for an alternative representation of gene regulatory networks [9]; it combines rule-based modeling with uncertainty principles. PBNs have been deployed in a spectrum of applications similar to those of BNs [10], [11].

For the past decade, BNs and PBNs have been the object of extensive studies. Past theoretical studies focus on learning [12]–[14], related to dynamic Bayesian networks [15], and Markov chains [8], [16]. Recent research efforts have primarily focused on their stability and control [17]–[28], synchronization [29]–[32], semitensor product (STP) representation [33], [34], and network identification [35]–[38]. Identification of the long-run behavior of dynamical systems is of utmost importance since it usually conveys domain implications. For instance, the characterization of the network’s long-run dynamics plays a crucial role in the treatment of various types of cancer such as breast cancer and leukemia [39]. Furthermore, it is possible to control certain nodes in a network, such that the whole system can evolve according to a desirable probability distribution [40]. However, automated system reconstruction remains very challenging, mainly due to scalability issues. (The exponential growth of possible model configurations and the limited observations under changing initial conditions make inference computationally infeasible for large networks.)

Our contributions revolve around model learning and dynamics inference for PBNs under an alternative representation. This formulation leads to the approximate maximum likelihood estimation (MLE) method which renders the tackling of both problems at a large-scale feasible. To the best of our knowledge, no prior research has managed to address the reconstruction of the state evolution of general PBNs on the order of 1000 nodes successfully (or equivalently a Markov Chain with $2^{1000}$ states).

II. RELATED WORK

As already discussed in Section I, the dynamical behavior of a PBN can be described by the Markov chain theory, and...
thus previously developed tools can be applied to the analysis of PBNs. An MLE approach for estimating the transition probability matrix of the Markov chain (and the associated PBN) is presented in [16], along with certain theoretical guarantees. However, straightforward transition probabilistic representations require the estimation of $2^N \times (2^N - 1)$ (or $2^N \times N$, in the case of independent nodes) probabilities, where $N$ is the number of nodes in the network. Therefore, such representations demand an unrealistic amount of data which hinders their adoption in real-world scenarios.

The learning procedure in [12] attempts to learn both the logical portion (in the form of a truth table) and the parametric portion (in the form of switching, selection, and perturbation probabilities). However, the amount of temporal data needed for estimating the parametric part, which is crucial for dynamics recovery, is huge. Indeed, only results for network connectivity are reported, and only for networks with no more than seven nodes. Moreover, a maximum, much smaller than $N$, number of different interacting nodes (node in-degree) in the Boolean functions known a priori are assumed, while the tabular representation of the learned PBN is unsuitable for further analysis. The approach described in [41] relies on prior domain knowledge in terms of the network’s biological function. The states of all nodes are updated synchronously at the same time. Let $F = \{f^{(1)}, f^{(2)}, \ldots, f^{(N)}\}$ be the network function. We denote the BN by $G(V,F)$.

Definition 1: A BN is a directed network with $N$ binary-valued nodes $V = \{x_1, x_2, \ldots, x_N\}$. Let $s_t(x) \in \{0,1\}$ be the state of node $x$ at time $t$, and $V(t) \subseteq V$ be the set of the parent nodes of node $x$. A BN is defined by

$$s_{t+1}(x) = f^{(i)}(s_t(x))_{x \in V(t)},$$

for $i = 1, 2, \ldots, N$ \hspace{1cm} (1)

where $f^{(i)} : \{0,1\}^{V(t)} \rightarrow \{0,1\}$ is a simplified Boolean function. The states of all nodes are updated synchronously (at the same time). Let $F = \{f^{(1)}, f^{(2)}, \ldots, f^{(N)}\}$ be the network function. We denote the BN by $G(V,F)$.

Definition 2: A PBN is also a directed network with $N$ binary-valued nodes $V = \{x_1, x_2, \ldots, x_N\}$. Let $s_t(x) \in \{0,1\}$ be the state of node $x$ at time $t$. In a PBN, a list $F^{(i)} = \{f_1^{(i)}, f_2^{(i)}, \ldots, f_{M(t)}^{(i)}\}$ of candidates for $f^{(i)}$ is provided for each $x_i$. Let $V(t) \subseteq V$ be the set of variables of function $f^{(i)}$. At each time point, one of these candidate functions is sampled independently, according to some distribution $p^{(i)} = \{P_1^{(i)}, P_2^{(i)}, \ldots, P_{M(t)}^{(i)}\} \in [0,1]^{M(t)}$, where $\sum_{j=1}^{M(t)} p_j^{(i)} = 1$. The PBN is then defined by

$$s_{t+1}(x_i) = f^{(i)}_{j_i}(s_t(x))_{x \in V(t)},$$

for $i = 1, 2, \ldots, N$ \hspace{1cm} (2)

where $j_i \sim$ Categorical($M(t)$, $p^{(i)}$). The states of all nodes are updated synchronously (at the same time) and independently. Let $F = \{F^{(1)}, F^{(2)}, \ldots, F^{(N)}\}$ be the network function, $P = \{p^{(1)}, \ldots, p^{(N)}\}$, and $\tilde{F}(F, P)$. We denote the PBN by $G(V,F)$.

A. Problem Statement

Assume that we observe $D_i$, a list of $T$ Boolean time series of potentially varying lengths $n_j$, for $j = 1, 2, \ldots, T$

$$D = ((S_{t_1}^1, S_{t_1}^1, \ldots, S_{t_1}^1), \ldots, (S_{t_T}^T, S_{t_T}^T, \ldots, S_{t_T}^T)) \hspace{1cm} (3)$$

where $S_{t_j}^j \in \{0,1\}^N$. $D$ can be transformed to a list (with repetition of elements) of ordered pairs $(S^j, S'^j)$ which represents the transition from the previous state $S$ to the next state $S'$ of the dynamical system

$$L \triangleq \bigcup_{j=1}^{T} \{(S_t^j, S'_{t+1}^j)\}_{t=0}^{T-1}. \hspace{1cm} (4)$$
Given the observed data $\mathbf{D}$, our goal is to infer the logical functions and the corresponding selection probabilities of the system, which best explain its behavior and can predict its evolution for different initial node states.

IV. PROPOSED MODEL

In SCNFNs, the rule that corresponds to each node is specified in conjunctive normal form (CNF), meaning that it consists of a conjunction (logical AND) of multiple disjunctions (logical OR) of Boolean variables. Stochasticity is induced at the level of each separate disjunction which is associated with a probability of being activated (evaluated), in which one literal is actually a Bernoulli random variable.

A. Definition of an SCNF Logical Clause

Definition 3: A stochastic CNF clause $\tilde{\Psi} : \{0, 1\}^N \rightarrow \{0, 1\}$ of $N$ variables $V = \{x_1, x_2, \ldots, x_N\}$ is defined as

$$\tilde{\Psi}(x_1, x_2, \ldots, x_N; \mathbf{p}) \triangleq \tilde{\psi}_1(x_{1,1}, x_{1,2}, \ldots, x_{1,N_1}; p_1) \wedge \tilde{\psi}_2(x_{2,1}, x_{2,2}, \ldots, x_{2,N_2}; p_2) \wedge \cdots \wedge \tilde{\psi}_M(x_{M,1}, x_{M,2}, \ldots, x_{M,N_M}; p_M)$$

(5)

where $\mathbf{p} = \{p_1, p_2, \ldots, p_M\}$ and $0 \leq p_j \leq 1$ for $j = 1, 2, \ldots, M$. Let $V_j = \{x_{j,1}, x_{j,2}, \ldots, x_{j,N_j}\} \subseteq V$ be the set of $N_j$ variables in clause $\tilde{\psi}_j$, where $V = \bigcup_{j=1}^M V_j$. Each clause $\tilde{\psi}_j$ in the conjunction $\tilde{\Psi}$ is defined as follows:

$$\tilde{\psi}_j(V_j; p_j) \triangleq \psi_j(V_j) \lor \neg \alpha_j$$

(6)

where

$$\psi_j(V_j) = l_{j,1} \lor \cdots \lor l_{j,N_j}$$

(7)

$$l_{j,n} \in \{x_{j,n}, \neg x_{j,n}\}$$

(8)

$$\alpha_j \sim \text{Bernoulli}(p_j)$$

(9)

for $n = 1, 2, \ldots, N_j$, $j = 1, 2, \ldots, M$, and where $\neg$ denotes the logical negation. $l_{j,n}$ refers to the $n$th literal (the node or its complement) of the $j$th disjunction in the SCNF clause. $\psi_j$ is a disjunction of literals, and $\tilde{\psi}_j$ is a stochastic disjunction: the disjunction $\psi_j$ augmented by the Bernoulli random variable $\alpha_j$.

By definition, the presence of the Bernoulli random literal $\alpha_j$ in the disjunction $\psi_j$ implies that with probability $1 - p_j$, $\psi_j$ does not contribute to the logical value of $\tilde{\Psi}$. This is because

$$\tilde{\psi}_j(V_j; p_j) = \begin{cases} 1, & \text{w.p. } (1 - p_j) \\ \psi_j(V_j), & \text{w.p. } p_j \end{cases}$$

(10)

where “w.p.” stands for “with probability.” The quantifier $n$ of literal $l_{j,n}$ refers to its lexicographic order within the disjunction $\psi_j$, such that $\neg x_j < x_j$, and $\neg x_j < \neg x_j$, $\neg x_j < x_j$, $x_j < \neg x_j$, and $x_j < x_j$ if $j < j'$, while the quantifier $j$ of disjunction $\tilde{\psi}_j$ refers to its lexicographic order in the SCNF rule $\tilde{\Psi}$: $\psi_j < \psi_j'$ if $\exists j \in \tilde{\psi}_j : l > l' \forall l'' : \{l'' \in \tilde{\psi}_j \land l'' \notin \tilde{\psi}_j'\}$.

B. Model Definition

Definition 4: A SCNFN is a directed network with $N$ binary-valued nodes $V = \{x_1, x_2, \ldots, x_N\}$. Let $s_i(t) \triangleq s_i(x_i) \in \{0, 1\}$ be the state of node $x_i$ at time $t$. This state is determined by the SCNF expression $\tilde{\Psi}(V(t); \mathbf{p}(t)) : \{0, 1\}^N \rightarrow \{0, 1\}$ such that

$$s_{i+1} = \tilde{\Psi}(s_i(x_i) \in V(t))$$

(11)

and $V(t) \subseteq V$ is the set of the $N(t)$ variables that determine the state of node $x_i$. The states of all nodes are updated synchronously (at the same time) and independently. Let $\Psi = [\tilde{\Psi}(1), \tilde{\Psi}(2), \ldots, \tilde{\Psi}(N)]$ be the network function. We denote the SCNFN by $G(V, \Psi)$.

The state of the whole network at time step $t$ is represented by the vector $S_t = [s_1(t), s_2(t), \ldots, s_N(t)] \in \{0, 1\}^N$. We denote the system transition after one time step of the SCNFN by $S_{t+1} = \tilde{\Psi}(S_t)$. It is implied that $\tilde{\Psi}(V(t); \mathbf{p}(t)) = \tilde{\Psi}(V(t); \mathbf{p}(t))$. Similarly, the dynamics after $\tau$ system transitions is represented as $S_{t+\tau} = \tilde{\Psi}^\tau(S_t)$.

C. Notational Conventions

We now list some notational conventions that we adopt for the rest of this paper.

1. We use superscript $(i)$ to refer to the SCNFN clause, disjunction or literal related to node $x_i$.
2. We use upper-case Greek letters $\Theta(i)$, $\Phi(i)$, and $\Psi(i)$ to represent the SCNFN clause related to node $x_i$.
3. We use lower-case Greek letters $\tilde{\Theta}_j$, $\tilde{\Phi}_j$, and $\tilde{\Psi}_j$ to represent the $j$th disjunction in the SCNFN rule of node $x_i$.
4. We may equivalently represent a conjunction $\tilde{\Theta}$ as a set of disjunctions and a disjunction $\tilde{\Theta}$ as a set of literals. Therefore, $|\tilde{\Theta}|$ is the number of disjunctions in the SCNFN rule $\tilde{\Theta}$, and $|\tilde{\Theta}|$ is the number of literals in the stochastic disjunction $\tilde{\Theta}$ [excluding the Bernoulli random variable defined in (9)].
5. We drop the tilde symbol from Greek letters to represent deterministic clauses. For example, if $p_j = 1.0$, we use $\psi_j$ instead of $\tilde{\psi}_j$. Similarly, we use $\psi_j$ instead of $\tilde{\psi}_j$. If $p_j = 1.0$, for $j = 1, 2, \ldots, |\tilde{\Psi}(i)|$.
6. We may drop the arguments from stochastic disjunctions or SCNFNs when they do not contribute to understanding the concepts being elaborated.
7. We use $v(i)$ to denote the $i$th element of the list $v$.

D. Equivalence of SCNFNs and PBNs

We now prove that any SCNFN can be converted into an equivalent PBN, and vice versa. This is expected since they both represent a discrete-time homogeneous Markov chain. This equivalence can also be viewed as the stochastic extension of the conversion of any propositional formula to the CNF.

Proposition 1: Every SCNFN $G(V, \Psi)$ can be converted to an equivalent PBN $G(V, F)$.

Proof: Clearly, $V = V$. Recall that $F = (F, P)$ and fix node $x_i$. We describe the conversion of the SCNFN rule $\tilde{\Psi}(i) = \tilde{\Psi}(i)$ to the list $f(i) = F(i)$ and the corresponding selection
probabilities \( p^{(i)} = P(i) \). First, we consider the decomposition \( \Psi^{(i)} = \Phi^{(i)} \land \Theta^{(i)}(q^{(i)}) \), where each disjunction \( \phi_j^{(i)} \in \Phi^{(i)} \) is deterministic, and each disjunction in \( \Theta^{(i)} \) is stochastic. By definition of the activation of the disjunctions in \( \Theta^{(i)} \), each rule \( f_j^{(i)} = f^{(i)}(j) \) corresponds to the logical AND of \( \Phi^{(i)} \) and an element of the power set which contains all possible subsets of the stochastic disjunctions in \( \Theta^{(i)} \). Let \( \beta_j \) be the \( j \)th element in this power set (also a SCNF clause) for \( j = 1, 2, \ldots, 2^{|\Theta^{(i)}|} \) (assuming a lexicographic ordering of the conjunctions). Then, the logic rule \( f_j^{(i)} \) is defined as
\[
 f_j^{(i)} = \Phi^{(i)} \land B_j(\tilde{\Theta}^{(i)}) \land \cdots \land B_j(\tilde{\Theta}_{\Theta^{(i)}}) \tag{12}
\]
where
\[
 B_j(\tilde{\Theta}_\zeta) = \begin{cases} 
 \tilde{\Theta}_\zeta, & \text{if } \tilde{\Theta}_\zeta \in \beta_j \\
 1, & \text{if } \tilde{\Theta}_\zeta \notin \beta_j
\end{cases} \tag{13}
\]
for \( \zeta = 1, 2, \ldots, |\tilde{\Theta}^{(i)}| \). The corresponding selection probabilities are defined as
\[
 p_j^{(i)} = p^{(i)}(j) = \prod_{\zeta=1}^{\tilde{\Theta}^{(i)}} (\tilde{\Theta}_\zeta)\mathbb{I} (\tilde{\Theta}_\zeta \in \beta_j) \left( 1 - \tilde{\Theta}_\zeta \mathbb{I} (\tilde{\Theta}_\zeta \notin \beta_j) \right) \tag{14}
\]
where \( \mathbb{I} \) is the numerical indicator function which returns 1 if the condition in its argument is true, and 0 otherwise. Equation (14) stems from the fact that the disjunctions in \( \tilde{\Theta} \) are assumed to be independently evaluated. This formulation constitutes a PBN.

Due to Proposition 1, there are \( M^{(i)} = 2^{|\tilde{\Theta}^{(i)}|} \) candidate Boolean functions for node \( x_i \), and \( M = \prod_{i=1}^{N} M^{(i)} \) constituent BNs, \( F_j = [f_1^{(i)}, f_2^{(i)}, \ldots, f_{M^{(i)}}^{(i)}] \), with corresponding selection probabilities \( p_j^{(i)} = \prod_{i=1}^{N} p^{(i)}_j \), where \( p_j^{(i)} \) is defined in (14). Therefore, the linear representation of SCNFNs can be easily obtained. A derivation is provided in the Supplementary Material.

**Proposition 2:** Every PBN \( G(V, F) \) can be converted to an equivalent SCNFN \( G(V, \Psi) \).

**Proof:** Clearly \( V = V \). Recall that \( \tilde{F} = (F, \Phi) \) and fix node \( x_i \). We describe the conversion of the list \( f^{(i)} = F(i) \) which contains all the \( M^{(i)} \) logic rules which regulate the dynamics of node \( x_i \) and the corresponding selection probabilities, \( p^{(i)} = P(i) \), to an equivalent SCNF rule \( \Psi^{(i)} = \Psi(i) \). \( \Psi^{(i)} \) consists of \( 2^N \) stochastic disjunctions \( \psi_j^{(i)} \), such that
\[
 \Psi^{(i)} = \psi_0^{(i)} \land \psi_1^{(i)} \land \cdots \land \psi_{2^N-1}^{(i)} \tag{15}
\]
and
\[
 \psi_j^{(i)}(x_1, \ldots, x_N; q_j^{(i)}) = \psi_j^{(i)}(x_1, \ldots, x_N; q_j^{(i)}) \lor \neg a_j^{(i)} \tag{16}
\]
for \( j = 0, 1, \ldots, 2^N - 1 \). \( \Psi^{(i)} \), thus, corresponds to each possible combination of the available literals \( L = \{ x_i, \neg x_i \}_{i=1}^{N} \) and each \( \psi_j^{(i)} \) can evaluate to 0 for exactly one network state.

Therefore,
\[
 \psi_0^{(i)} = \neg x_N \lor \cdots \lor \neg x_2 \lor \neg x_1 \\
 \psi_1^{(i)} = \neg x_N \lor \cdots \lor \neg x_2 \lor x_1 \\
 \vdots \\
 \psi_{2^N-1}^{(i)} = x_N \lor \cdots \lor x_2 \lor x_1. \tag{17}
\]

To find parameter \( q_j^{(i)} \), note that \( q_j^{(i)} \) is equivalent to the probability that the SCNF rule \( \psi_j^{(i)} \) will evaluate to 0 for state \( \lambda_j^{(i)} \), because the rest of the disjunctions will always go to 1 (no matter what the outcome of their associated Bernoulli variables is). Therefore, they have no effect on the evaluation of the system for state \( \lambda_j^{(i)} \).

Subsequently, we compute the probability that the PBN will evaluate to 1 for state \( \lambda_j^{(i)} \), given the constituent Boolean rules \( f_j^{(i)} \) and their corresponding selection probabilities \( p_j^{(i)} \). This is equal to the probability that a logical rule which yields 1 for state \( \lambda_j^{(i)} \) will be selected from \( f^{(i)} \). Therefore, the parameter of each Bernoulli variable \( q_j^{(i)} \) of the jth stochastic disjunction can be defined as
\[
 q_j^{(i)} = 1 - \sum_{j=1}^{M^{(i)}} p_j^{(i)} f_j^{(i)}(\lambda_j^{(i)}). \tag{18}
\]

In the Supplementary Material, we provide an example which illustrates this proof using a three-node network.

**V. Learning SCNF Networks**

We now describe the general idea and the main components involved in learning an SCNFN. SCNFN learning amounts to a statistical process which can be viewed as a “logic” regression problem; the entries of the transition probability matrix (i.e., dependent variables) are estimated using the Boolean relationships and a Boolean representation for the system states (i.e., independent variables). This process entails a significant reduction in the number of parameters to be estimated. Therefore, in contrast to the prior work, SCNFNs can be learned for large networks, from small training data sets. The learning algorithm greedily—as opposed to optimally—maximizes the likelihood of the training data. Finding the optimal solution is a combinatorial problem, which prevents any exact algorithm from being practically applicable for large-scale network structures. Our methodology consists of two parts:

1) learning the logical interactions between nodes in the system (CNFLogicLearn, shown in Algorithm 2 and CNFDisjunctionLearn, shown in Algorithm 3);
2) learning the parameters of the Bernoulli random variables associated with the disjunctions discovered in the previous step (Lines 17–20, Algorithm 1).

Each invocation of Algorithm 1 learns the SCNF clause of a single node \( x_i \). Given list \( L \), defined in (4), the reduced list \( L^{(i)} \), which holds pairs of the previous system state and node \( x_i \)'s next state is formed, as follows:
\[
 L^{(i)} \doteq \{(S, s^{(i)}) : (S, S') \in L\}. \tag{19}
\]
Algorithm 1 SCNFLearn

1: Inputs
2: $L^{(i)}$: The list of transitions of a node $x_i$ (Equation 19).
3: $L = \{x_i, \neg x_i\}_{i=1}^{N}$: The set of the available literals.
4: Output
5: $\tilde{\Psi}^{(i)}$: The SCNF clause of the node $x_i$.
6: Begin
7: Form $S_0^{(i)}, S_1^{(i)},$ and $S_C^{(i)}$ (Equations (20), (21), and (22)).
8: If $S_0^{(i)} = \emptyset$
9: $\phi^{(i)} \leftarrow 1$
10: Else If $S_0^{(i)} \cup S_C^{(i)} = \emptyset$
11: $\phi^{(i)} \leftarrow 0$
12: Else
13: $\phi^{(i)}(V_d^{(i)}) \leftarrow$ CNFLogicLearn($S_0^{(i)}, S_1^{(i)} \cup S_C^{(i)}, L$), $p_d^{(i)} \leftarrow 1$.
14: End If
15: If $S_0^{(i)} \neq \emptyset$
16: $\tilde{\phi}^{(i)}(V_s^{(i)}) \leftarrow$ CNFLogicLearn($S_C^{(i)}, S_1^{(i)}, L$)
17: CNFParameterLearn($L^{(i)}, S_1^{(i)}, \phi^{(i)}$);
18: $\forall \lambda \in S_C^{(i)}$ Form $\tilde{\psi}^{(i)}_0(\lambda)$ (Equation 23).
19: Find optimization problem, $OP$, and solve it
20: $(p_i, p_0^{(i)}(\lambda))_{\lambda \in S_C^{(i)}} \leftarrow$ conSolve($OP$)
21: Else
22: $\tilde{\phi}^{(i)} \leftarrow 1$
23: End If
24: $\tilde{\Psi}^{(i)}(V_d^{(i)} \cup V_s^{(i)}; p_d^{(i)} \cup p_i^{(i)}) \leftarrow \Phi^{(i)}(V_d^{(i)}; p_d^{(i)}) \land \tilde{\phi}^{(i)}(V_s^{(i)}; p_i^{(i)})$
25: Return $\tilde{\Psi}^{(i)}$

In the preprocessing step of Algorithm 1, the structure $L^{(i)}$ is parsed and the sets $S_0^{(i)}, S_1^{(i)},$ and $S_C^{(i)}$ are formed (Line 7, Algorithm 1). The set $S_0^{(i)}$ consists of the states in $L^{(i)}$ which steer node $x_i$ only to 0

$$S_0^{(i)} \triangleq \{S : (S, 0) \in L^{(i)}, (S, 1) \notin L^{(i)} \}.$$  

Similarly, the set $S_1^{(i)}$ consists of the states in $L^{(i)}$ which steer node $x_i$ only to 1

$$S_1^{(i)} \triangleq \{S : (S, 1) \in L^{(i)}, (S, 0) \notin L^{(i)} \}.$$  

Finally, the set $S_C^{(i)}$ contains the system states which drive node $x_i$ both to a 0 and a 1 state in $L^{(i)}$

$$S_C^{(i)} \triangleq \{S : (S, 1) \in L^{(i)}, (S, 0) \in L^{(i)} \}.$$  

Note also that the set of all different system states is the same for all nodes $i = 1, 2, \ldots, N$; $S \triangleq S_C^{(i)} \cup S_0^{(i)} \cup S_1^{(i)}$.

The learning of SCNF formulas (Algorithm 1) consists of four main parts.

1) Lines 8–14: The deterministic logical portion $\Phi^{(i)}$ of the SCNF rule is learned. In this part, all the disjunctions are always evaluated. (The corresponding parameter is 1.) CNFLogicLearn (Algorithm 2) is responsible for returning a CNF rule which evaluates to 0 for all transitions passed in its first argument and 1 for all transitions contained in its second argument. Note here that the transitions in $S_C^{(i)}$ are treated as positive, since we want to avoid them always being evaluated to 0 by $\phi^{(i)}$.

2) Line 16: The stochastic portion $\tilde{\phi}^{(i)}$ of the SCNF rule is learned. The transitions in $S_0^{(i)}$ can now be ignored, because there exists at least one disjunction in $\phi^{(i)}$ which evaluates to 0 when the previous state in $S_0^{(i)}$ is substituted in its variables $V_d^{(i)}$. On the other hand, the transitions in $S_C^{(i)}$ are now treated as negative by $\tilde{\phi}^{(i)}$, because there should be at least one disjunction which probabilistically turns to 0 when a previous state in $S_C^{(i)}$ is plugged in its variables $V_s^{(i)}$ (otherwise, the transition would deterministically evaluate to 1). The transitions in $S_1^{(i)}$ are treated as positive by $\tilde{\phi}^{(i)}$, as well, so that the concatenated proposition $\tilde{\Psi}^{(i)} = \phi^{(i)} \land \tilde{\phi}^{(i)}$ deterministically evaluates to 1.

3) Lines 17–20: The parametric part of $\tilde{\phi}^{(i)}$ and $p_i^{(i)}$ is learned, by solving an optimization problem.

4) Line 24: The deterministic portion, $\Phi^{(i)}$, and the stochastic portion, $\tilde{\phi}^{(i)}$, are merged into the full SCNF rule, $\tilde{\Psi}^{(i)}(V_d^{(i)}; p_i^{(i)})$ for node $x_i$.

Algorithm 2 CNFLogicLearn

1: Inputs
2: $H_0$: The set of negative transitions.
3: $H_1$: The set of positive transitions.
4: $L$: The set of the available literals.
5: Output
6: $\Phi$: A CNF clause, for which $\Phi(S) = 0, \forall S \in H_0$, and $\Phi(S) = 1, \forall S \in H_1$
7: Begin
8: $k \leftarrow 0, \Phi \leftarrow \emptyset$
9: Repeat
10: $\phi_k \leftarrow$ CNFDisjunctionLearn($H_0, H_1, L, \emptyset$)
11: $H_0 \leftarrow H_0 - \{S : S \in H_0, \phi_k(S) = 0 \}$
12: $\Phi \leftarrow \Phi \land \phi_k$
13: $k \leftarrow k + 1$
14: Until $H_0 = \emptyset$
15: Return $\Phi$

The routine conSolve refers to TOMLAB’s “conSolve” solver for general, constrained, and nonlinear optimization problems.
A. Logic Learning

We now delve into the details of learning the logic rules (Algorithms 2 and 3). Algorithm 2 returns a CNF rule which evaluates to 0 for all states in $H_0$ and to 1 for all the states in $H_1$ (Line 6). This implies that each disjunction in $\Phi$ should evaluate to 1 for all states in $H_1$ and that for each state in $H_0$, there should be at least one disjunction in $\Phi$ which evaluates to 0. Each loop iteration in Algorithm 2 adds a new disjunction $\phi_i$ in the currently formed conjunction $\Phi$ (Line 12). The function $\text{CNFDisjunctionLearn}$ is responsible for returning a disjunction which satisfies two conditions: 1) it evaluates to 0 for at least one state passed in its first argument and 2) it evaluates to 1 for all states passed in its second argument. This guarantees that the condition in Line 14 will eventually become true and the algorithm will terminate.

If a transition in $H_0$ evaluates to 0 due to disjunction $\phi_i$, the transition is removed (Line 11), because this suffices for the whole $\Phi$ to yield 0. Note that $H_1$ does not change because each disjunction $\phi_i$ should evaluate to 1, for each transition in $H_1$, in order for the final CNF clause $\Phi$ to be in accordance with both $H_0$ and $H_1$. Once $H_0$ becomes empty (i.e., all the negative transitions are satisfied by $\Phi$) (Line 14, Algorithm 2), the algorithm terminates.

B. Disjunction Learning

Each recursive call of Algorithm 3 inserts a new literal $l^*$ in the current disjunction $\phi$ (see Lines 30 and 33). The inclusion of the new literal $l^*$ results in some transitions in $H_1$, as well as in $H_0$, turning to 1. This is a desirable change for $H_1$, since it reduces the number of recursive calls required for the new disjunction to satisfy the first condition in Line 7, as guaranteed by Line 29. On the other hand, for $H_0$, this change is not desirable. This is because it decreases the chances of the second condition in Line 7 becoming satisfied, and it increases the required number of disjunctions in the final SCNF rule so that the condition in Line 15, Algorithm 2 is satisfied.

Therefore, for each candidate literal, we compute a positive score (see Line 18), according to the number of positive transitions that will be satisfied if the literal is included in the disjunction and a respective negative score (see Line 19). The literal with the maximum normalized score (Line 20) is selected (Line 23). In Lines 26–28, we check whether the selected literal has nonpositive score. If this is the case, the recursion has to stop since the inclusion of the new literal in $\phi$ does not contribute to the fulfilment of the first condition in Line 7. Similarly, if after the inclusion of the new literal, all the transitions in $H_0$ evaluate to 1 due to $\phi \lor l^*$, then the current branch of the recursion leads to an invalid disjunction. In both cases, the algorithm continues to find the next best literal (Lines 27 and 35), by excluding $l^*$ from the set of the available literals that can be used for the formation of the remaining disjunction. In case the inclusion of $l$ still preserves the validity of the disjunction $\phi \lor l^*$, $H_1$ and $H_0$ are reduced in Lines 24 and 25, and the recursion proceeds by removing both $l^*$ and $\neg l^*$ from the set of the available literals (Line 33). If the condition in Line 29 becomes true, a valid disjunction is discovered and returned. Otherwise, the algorithm proceeds to a recursive call with the updated literal sets. In the worst case, the algorithm will perform exhaustive search in the space of all possible disjunctions until it discovers a valid formula. However, due to the use of heuristic scores for literals (Line 20), the number of backtracking steps does not affect the computational performance of the algorithm. Lines 13 and 14 refer to the degenerate case where there is no negative transition, and therefore a single disjunction will be returned by Algorithm 2. Lines 15 and 16 refer to the degenerate case where there is no positive transition, and therefore a conjunction of literals will be returned by Algorithm 2.

C. Parameter Learning

In the last step of the learning procedure, the parameters $p_j^{(i)}$ of the stochastic disjunctions $\phi_j^{(i)}$ in the SCNF rule of
node $x_i$ are estimated, so that the likelihood of the time series [transitions in $L^{(i)}$, (19)] is maximized. Let us first introduce some new notation that will be useful in describing this part of the system.

Let $\tilde{\Psi}^{(i)}(\lambda)$ be the set of disjunctions in $\tilde{\Psi}^{(i)}$ which can evaluate to 0 (given that they are activated, i.e., $a^{(i)}_j = 1$) for state $\lambda$

$$\tilde{\Psi}^{(i)}(\lambda) \triangleq \{ \tilde{\psi}^{(i)}_j : \psi^{(i)}_j \in \tilde{\Psi}^{(i)}, \psi^{(i)}_j(\lambda) = 0 \}.$$  

(23)

The above definition considers the case when $a^{(i)}_j = 1$; hence, $p^{(i)}_j$ is intentionally omitted in the arguments of $\psi^{(i)}_j$.

Let $P^{(i)}_0(\lambda)$ represent the probability that the previous state $\lambda$ of the system will switch node $x_i$ to 0 in the next step. Formally,

$$P^{(i)}_0(\lambda) \triangleq \text{Probability}[\tilde{\Psi}^{(i)}(\lambda; p^{(i)}) = 0].$$

(24)

This quantity can be computed using the inclusion–exclusion principle [43] on the activation of at least one disjunction in $\tilde{\Psi}^{(i)}(\lambda)$

$$P^{(i)}_0(\lambda) = \sum_{m=1}^{S} (-1)^{m+1} \sum_{\bar{\theta}_{jm} \in (\tilde{\Psi}^{(i)}_m(\lambda))} \prod_{n=1}^{m} p^{(i)}_{jm(n)}$$

(25)

where $(S)_m$ denotes the set of all subsets of set $S$ of cardinality $m$. $\bar{\theta}_{jm}$ denotes the $jm$th element/SCNF clause in $(\tilde{\Psi}^{(i)}_m(\lambda))$ (while assuming a lexicographic ordering of the SCNFs) and is defined as

$$\bar{\theta}_{jm} = \tilde{\psi}^{(i)}_{j1} \land \tilde{\psi}^{(i)}_{j2} \land \cdots \land \tilde{\psi}^{(i)}_{jm(m)}$$

(26)

where $\tilde{\psi}^{(i)}_{jm(n)}$ is $n$th stochastic disjunction in the $jm$th SCNF formula (while also assuming a lexicographic ordering), such that $jm(n) = j$ for some $j$ for which $\tilde{\psi}^{(i)}_j \in \tilde{\Psi}^{(i)}(\lambda)$.

We also define $N_0^{(i)}(\lambda)$ and $N_1^{(i)}(\lambda)$, which are the number of times the system state $\lambda$ drives node $x_i$ to a low state and a high state, respectively, in the list $L^{(i)}$

$$N_0^{(i)}(\lambda) \triangleq |\{ S : (S, s) \in L^{(i)}, S = \lambda, s = 0 \}|$$

(27)

and

$$N_1^{(i)}(\lambda) \triangleq |\{ S : (S, s) \in L^{(i)}, S = \lambda, s = 1 \}|.$$ 

(28)

Then, the likelihood of the time series $L^{(i)}$ is defined as

$$L_i(L^{(i)}; p^{(i)}) = \prod_{\forall \lambda \in \mathbb{S}_C^{(i)}} P^{(i)}_0(\lambda)^{N_0^{(i)}(\lambda)} (1 - P^{(i)}_0(\lambda))^{N_1^{(i)}(\lambda)}.$$ 

(29)

In order to derive (29), we may consider each of the $N_0^{(i)}(\lambda) + N_1^{(i)}(\lambda)$ appearances of state $\lambda$ as a realization of a binomial process which yields 0 as the next state for node $x_i, N_0^{(i)}(\lambda)$ times, and 1, $N_1^{(i)}(\lambda)$ times. Moreover, only the transitions in $\mathbb{S}_C^{(i)}$ contribute to the estimation of $p^{(i)}$ since $\forall \lambda \in \mathbb{S}_0^{(i)} : P^{(i)}_0(\lambda) = 1.0$ and $\forall \lambda \in \mathbb{S}_1^{(i)} : P^{(i)}_0(\lambda) = 0.0$ (due to the constraints in the construction of the logical rules). Therefore, it holds that $L_i(L^{(i)}; p^{(i)}) = L_i(L^{(i)}; p^{(i)}_0)$, where $p^{(i)}_0 = (p^{(i)}_j : p^{(i)}_j \in p^{(i)}, p^{(i)}_j \neq 1.0$ from Line 13 in Algorithm 1). Subsequently, by taking the natural logarithm of (29), we obtain the log-likelihood

$$l_i(L^{(i)}; p^{(i)}) \triangleq \log L_i(L^{(i)}; p^{(i)}).$$

(30)

We can now define an optimization problem to find the value of $p^{(i)}$ which maximizes the likelihood of the transitions in the training set $L^{(i)}$ (or equivalently minimizes the negative log-likelihood):

$$\min_{\forall \lambda \in \mathbb{S}^{(i)}_C} \left\{ - \sum_{\forall \lambda \in \mathbb{S}^{(i)}_C} l_i(\lambda) \right\} \text{ over } p^{(i)} \cdot \{P^{(i)}_0(\lambda)\}_{\forall \lambda \in \mathbb{S}^{(i)}_C}$$

(31)

subject to

$$0 \leq p^{(i)}_j \leq 1, \quad p^{(i)} \in \mathbb{R}^{(\tilde{\Theta}^{(i)})}$$

(33)

and

$$\forall \lambda \in \mathbb{S}^{(i)}_C : P^{(i)}_0(\lambda) = \sum_{m=1}^{(S)} (-1)^{m+1} \sum_{\bar{\theta}_{jm} \in (\tilde{\Psi}^{(i)}_m(\lambda))} \prod_{n=1}^{m} p^{(i)}_{jm(n)}$$

(34)

The constraints of (33) are necessary to ensure that the optimization variables $p^{(i)}$ refer to the activation probabilities of the stochastic disjunctions in $\tilde{\Theta}^{(i)}$, for the learned clause of node $x_i$. The auxiliary optimization variables $P^{(i)}_0(\lambda)$ connect the likelihood of the conflicting transitions with the activation probabilities.

Note that it is very likely that the feasible region of the optimization problem as defined in (31)–(34) is the empty set (due to the constraints between the optimization variables $P^{(i)}_0(\tilde{\Theta}^{(i)}(\lambda)$ and $p^{(i)}$ in (34) which may define an overdetermined system). One way to resolve this problem is to relax the constraints as follows:

$$\min_{\forall \lambda \in \mathbb{S}^{(i)}_C} \left\{ \sum_{\forall \lambda \in \mathbb{S}^{(i)}_C} l_i(\lambda) + \lambda \sum_{\forall \lambda \in \mathbb{S}^{(i)}_C} \varepsilon_i^2(\lambda) \right\} \text{ over } p^{(i)} \cdot \{P^{(i)}(\lambda)\}_{\forall \lambda \in \mathbb{S}^{(i)}_C}$$

(35)

where

$$\varepsilon_i(\lambda) = p^{(i)}_0(\lambda) - \sum_{m=1}^{(S)} (-1)^{m+1} \sum_{\bar{\theta}_{jm} \in (\tilde{\Psi}^{(i)}_m(\lambda))} \prod_{n=1}^{m} p^{(i)}_{jm(n)}$$

(36)

subject to

$$0 \leq p^{(i)}_j \leq 1, \quad p^{(i)} \in \mathbb{R}^{(\tilde{\Theta}^{(i)})}$$

(37)

for $\lambda \geq 0$. The term in (36) is the error induced by the relaxation of the constraint defined in (34). (A similar relaxation of
probability constraints was used in [44].) We solve the problem for multiple values of λ in order to avoid local optima. Our implementation uses the TOMLAB base module “conSolve” solver, which is standard for solving general, constrained, and nonlinear optimization problems.

VI. ILLUSTRATIVE EXAMPLE: LEARNING A SIMPLE SCNF

Consider a set D which consists of a single time series with the 10 transitions provided in Table I. The list \( L^{(A)} \) for node A and the sets \( S_0^{(A)} \), \( S_1^{(A)} \), and \( S_2^{(A)} \) are defined as

\[
L = \{(S_0, S_1), (S_1, S_2), (S_2, S_3), (S_3, S_4), (S_4, S_5), (S_5, S_6), (S_6, S_7), (S_7, S_8), (S_8, S_9), (S_9, S_{10}) \}
\]

\[
L^{(A)} = \{(S_0, 0), (S_1, 0), (S_2, 1), (S_3, 0), (S_4, 1), (S_5, 0), (S_6, 0), (S_7, 1), (S_8, 1), (S_9, 1), (S_{10}, 0) \}
\]

\[
S_0^{(A)} = \{\lambda_1, \lambda_3, \lambda_5, \lambda_6\}
\]

\[
S_1^{(A)} = \{\lambda_2, \lambda_4, \lambda_7, \lambda_8\}
\]

\[
S_2^{(A)} = \{\lambda_0\}.
\]

Note that transition \( \lambda_0 \) is conflicting, since at times 1 and 11, node A has state 0 for the system state \( \lambda_0 \), while at time 9, it has state 1. Table II provides the scores of the literals for each step of the learning process. We now provide some detail for the computation of the score of \( \neg B \) in the first step: 1) score\(^{+}\)(\( \neg B \)) = 3, for states \( \lambda_0, \lambda_7, \) and \( \lambda_8 \); 2) \( |H_1| = |S_2^{(A)}| + |S_1^{(A)}| = 4 + 1 = 5 \); 3) score\(^{-}\)(\( \neg B \)) = 0; 4) \( |H_0| = 4 \); and 5) score(\( \neg B \)) = (3/5) − (0/4). The step-by-step construction of the SCNF rule can be found in Table III. Note that \( \lambda_0 \) is initially in \( H_1 \) and, once the reconstruction of the stochastic part starts at Step 4, it switches to \( H_0 \). After the inclusion of the first literal \( \neg B \), the states \( \lambda_0, \lambda_7, \) and \( \lambda_8 \) are removed from \( H_1 \) because the disjunction evaluates to 1 due to the presence of \( \neg B \). On the other hand, \( H_0 \) remains the same since \( \neg B \) does not switch to 1 for any transition contained in it. After the inclusion of literal \( \neg A \), all transitions in \( H_1 \) evaluate to 1, and the algorithm proceeds to the formation of the next disjunction. The disjunction \( (\neg B \vee \neg A) \) yields 0 for states \( \lambda_3, \lambda_5, \) and therefore the next disjunction should give 0 only for states \( \lambda_1 \) and \( \lambda_6 \). Learning of the deterministic part finishes when \( H_0 \) becomes empty. Steps 5 and 6 pertain to learning the stochastic part, which proceeds in the same way once the new sets, \( H_1 \) and \( H_0 \), are formed. Step 7 in Table III refers to learning the parametric part of the SCNF model. The relevant quantities are: \( N_0(\lambda_0) = 2, N_1(\lambda_0) = 1, N_2(\lambda_0) = \{\neg A \vee \neg E \vee \neg \alpha_1^{(A)}\}, p_0(\lambda_0) = p_1(\lambda_0), v_0(\lambda_0) = p_2(\lambda_0), \) \( \alpha_4(\lambda_0) = 0, \) \( \lambda_4(\lambda_0) = 2 \ast \log(p_1^{(A)}) + 1 \ast \log(1 - p_1^{(A)}), \) and \( p_1^{(A)} = 0.67 \).

VII. EXPERIMENTAL RESULTS—SYNTHETIC NETWORKS

In this section, we evaluate the predictive capability of the proposed learning algorithm. Before going into details, we would like to point out the weaknesses of existing methods, in which we are able to handle better. The BESTFIT-Extension [45] is considered to be state of the art for learning probabilistic Boolean relationships from stochastic data [46, 47]. If more than one BNs satisfy the time series data with the same minimum error, then the candidate BN is uniformly selected. Moreover, a maximum node in-degree of \( k \) is assumed and defined \( a \) \textit{a priori}. The method becomes computationally prohibitive for large \( k \) and \( N \). This is because it performs exhaustive search in the space of all \( 2^k \) possible Boolean formulas with \( k \) variables, in order to identify the formulas with minimum error. Consequently, it has complexity \( O((2^k) \times \binom{N}{k} \times N \times |S|) \), where \( |S| \) is the total number of system states present in the training set [45]. Therefore, computation time quickly explodes for large \( k \) and \( N \). In addition, small \( k \) values can be very restrictive for networks with very low connectivity such as gene regulatory networks, if temporal delays [48] have to be considered, or for complex...
systems such as cell signaling networks, where the number of interacting features can be large.

The method proposed in [12] does learn the parametric portion of the PBN in terms of switching, selection, and perturbation probabilities. However, it only manages to reconstruct the PBN of a seven-node network from an unrealistically large amount of training data. Specifically, predictive capacity emerges after training with a single temporal sequence of \( \sim 20,000 \) transitions, which is much larger than the data complexity required for the almost perfect reconstruction of a 10-node network with our method (20 time series of 16 time steps—the network was tested on all \( 2^{10} \) different initial states). This intractability is due to the tabular representation of the transition probability matrix that the authors use. Therefore, for large \( N \), the \( 2^N \) rows, which correspond to the previous state of the system, cannot be sufficiently populated. As expected, for states that have not been encountered while training, the method yields poor predictive performance, and, most importantly, this is inevitable for large networks. Finally, another source of degenerated accuracy and computational complexity is the fact that as before, a set of at most \( k \) influence variables is searched exhaustively.

In contrast, our proposed model, the SCNFN, uses Boolean logic to approximate the transition probabilities. The assumption on the independence of the stochastic disjunctions in SCNFNs can be viewed as an analog of the Naive Bayes assumption [49], which helps reduce the number of parameters required for estimating the joint probability distribution of a set of random variables. In the case of PBNs, according to the SCNFN formalism, the set of parameters that has to be estimated is reduced to the set of activation probabilities of the stochastic disjunctions. Furthermore, the candidate variables are selected greedily based on their score (Line 23, Algorithm 3). The hypothesis space is infinite and equivalent to the set of all possible PBNs with \( N \) nodes. To the best of our knowledge, no prior work has been able to handle the full reconstruction of general PBNs (both the structural and parametric parts) without prior knowledge being provided or structural constraints being enforced. More importantly, our approach significantly supersedes previous work, both in terms of computation time, but also by being the only approach to achieve very high accuracies for large networks.

A. Recovery of Time Homogeneous Markov Chain Dynamics

To assess the similarity between the target and the learned model, we consider the following setting:

1) We draw random initial system states \( \{\lambda_j\}_{j=1}^N \).

2) We simulate the target and the learned model \( R \) times for each \( \lambda_j \) in order to obtain \( \{s'_r(x_j|\lambda_j)\}_{r=1}^R \) and \( \{\hat{s}'_r(x_j|\lambda_j)\}_{r=1}^R \). \( s'_r(x_j|\lambda_j) \) and \( \hat{s}'_r(x_j|\lambda_j) \) are the states of node \( x_j \) at time \( r \), given initial system state \( \lambda_j \), for the \( r \)th stochastic run of the target and learned model, respectively.

3) We compute the sample estimates of the probability that node \( x_j \) will be in state \( 1 \) after \( \tau \) steps, given initial state \( \lambda_j \) in the target and learned model, respectively,

\[
P_l^{(i)}(\lambda_j, \tau) = \frac{1}{R} \sum_{r=1}^R s_r'(x_j|\lambda_j) \tag{38}
\]

\[
\hat{P}_l^{(i)}(\lambda_j, \tau) = \frac{1}{R} \sum_{r=1}^R \hat{s}_r'(x_j|\lambda_j). \tag{39}
\]

Note that for large \( \tau \) and \( N \), it is computationally intractable to obtain a closed-form value.

4) We compute the prediction error, as follows:

\[
\delta_j^{(i)}(\tau) = |P_l^{(i)}(\lambda_j, \tau) - \hat{P}_l^{(i)}(\lambda_j, \tau)|. \tag{40}
\]

Subsequently, we compute statistics for these differences across all nodes and all different initial states

\[
\bar{\delta}(\tau) = \frac{1}{N} \sum_{j=1}^N \delta_j^{(i)}(\tau), \quad \bar{\delta}(\tau) = \frac{1}{I} \sum_{j=1}^I \delta_j(\tau) \tag{41}
\]

\[
\sigma(\tau) = \frac{1}{I} \sum_{j=1}^I (\bar{\delta}(\tau) - \bar{\delta}(\tau))^2. \tag{42}
\]

The standard deviation across the different initial states lets us evaluate the dependence of the model’s performance on the initial states of the system.

The quantity \( N \times \bar{\delta}(\tau) \) can be interpreted as the average number of wrongly predicted states at time \( \tau \), regardless of the initial state. Alternatively, other metrics such as the Kullback–Leibler divergence and the Hellinger distance [43] were investigated. However, we only report the absolute difference given its direct and intuitive interpretation. Note that only the initial state is provided to the network. After the first step, the predicted states are used for the prediction of the next step states.

1) Learning a Small, Reference Network: In order to demonstrate the validity of the proposed model and learning algorithm, we use our method to learn the mammalian cell cycle; a 10-node network introduced by Faure [50] for which state-of-the-art prior work has been effective [47]. It is worth noting that the training data series are noisy, thus allowing us to test the robustness of the learning method. All of the possible \( 2^{10} \) time sequences were used for testing. We show the metrics defined in (41) and (42) in Fig. 1. Some wrongly predicted attractor states are the most likely cause for the oscillation of the error when the network is recovered using only 20 sequences. This result indicates that SCNFLearn is able to perfectly reconstruct the reference network using \( T = 40 \) time sequences and three possible system transitions. The small variance in Fig. 1(b) is due to the noisy behavior of the network. The network structure was also perfectly reconstructed, as expected due to the exact dynamics recovery (for more details, refer to Section IX).

2) Learning Large Networks: We also evaluate the learning algorithm for a 100-node and a 1000-node model. The target PBNs were generated as follows.

1) We used the “generateRandomNKNetwork” function provided by the BoolNet R-tool [47] to randomly generate two deterministic Boolean Networks in a disjunctive normal form representation. The deterministic BNs
have homogeneous topologies\(^2\) and uniform linkage\(^3\).

The behavior of all nodes in the resulting network is governed by two Boolean rules, which are provided by the constituent BNs.

2) We randomly merged the BNs into a PBN by assigning a selection probability to each rule. The selection probability among the rules of the stochastic nodes follows a uniform distribution over the [0, 1] interval.

To obtain the time series required for learning models and evaluating them, we simulated the true models for a variety of initial states, a number of times for the same initial state, and a number of time steps for each model size. The final model was learned by using fivefold cross-validation [49], where each fold contains 1/5 of the total number, \(T\), of the training time series. The 100-node and 1000-node models were evaluated on \(I = 10^4\) different initial states. For each one of them, 400 stochastic simulations were performed (i.e., \(R = 400\)). In all cases, the testing time series contain 1000 time points. The initial states of both the generated training and testing time series were randomly sampled.

The predictive capacity achieved by the learned models for various training data sets \(D^4\) is depicted in Figs. 2 and 3. The black line shows the baseline unavoidable error, which occurs during the simulation of the true model, because of using a finite value for \(R\) for the estimation of \(P_{ij}^{(1)}(\kappa, \tau)\) in (38). In order to compute the unavoidable error, a second round of \(R\) stochastic simulations of the real model for the initial states \(\kappa_j^{(1)}\) was performed. It is clear that the SCNFN manages to capture, almost perfectly, the dynamical behavior of the true PBN for \(N = 100\) [the black and green lines shown in the Fig. 2(a) almost overlap] with low standard deviation for the miscalculation rate, when using randomized initial states for testing [black and green lines in the Fig. 2(b)]. In the 1000-node case, the best average number of miscalculation nodes in all time points is roughly \(0.07 \times 1000\) green line shown in Fig. 3(a) compared to the performance of the true model, which is \(0.03 \times 1000\) black line shown in Fig. 3(a). However, as it is corroborated by the blue and pink lines shown in Fig. 3(a) and (b), a network learned with only 125 single-step time series can still exhibit high accuracy. We also proceeded to the reconstruction of a \(10^4\)-node network which achieved an error rate roughly 0.1 for \(I = 100\). However, due to limited computational resources, we were unable to test its performance for larger \(I\); therefore, we omitted these results. Finally, we checked the robustness of the learning algorithm with respect to the Boolean sequences given to the learner. To this end, we repeated the experiments with the same testing Boolean sequences but for different, uniformly drawn training sets of the same sizes as described in Figs. 2 and 3. We achieved very similar accuracy.

All tests are run on a Linux machine (Ubuntu 14.04 LTS) with CPU as Intel i7-5820K and 32GB DDR4 memory. Table IV demonstrates the runtime demands of the learning algorithm (for learning the five networks for the cross-validation). It should be mentioned that the CPU time does not include the time required for parsing the training time series and for the formation of the data structures \(L^{(1)}, S_0^{(1)}, S_1^{(1)}, S_{00}^{(1)}\). It pertains to the time required for the execution of Lines 13 (deterministic logic learning), 16 (stochastic logic learning), 17 (parameter learning) of Algorithm 1. In Table IV, we also provide the CPU demands of the BESTFIT-Extension method. On comparison, SCNFLearn outperforms BESTFIT-Extension for all cases of training data sizes and \(N = 100\) even when parameters were as small as \(k = 4\) (\(k\) being the maximum node in-degree). For the 100-node network, BESTFIT-Extension only reaches an accuracy close to (but still less than) SCNFLearn for \(k = 5\) (see Fig. 4), and even under those circumstances, SCNFLearn is the fastest method by 2 orders of magnitude. If BESTFIT-Extension does...
TABLE IV
SCNFN AND BESTFIT-EXTENSION LEARNING TIME FOR THE PBNs OF FIGS. 2 AND 3. THE SPEEDUP OF SCNFLearn OVER BESTFIT-EXTENSION IS ALSO DENOTED INSIDE THE PARENTHESIS.

| Network Size | Sample Complexity | Deterministic Learning Time (s) | Stochastic Learning Time (s) | Parameter Learning Time (s) | Total Learning Time (s) | Speedup
|--------------|--------------------|-------------------------------|-----------------------------|---------------------------|-------------------------|--------|
| 100          | 100/10/2           | 0.190 (0.013X)               | 0.806 (0.071X)              | 38.060 (3.27X)           | 1243.39 (107.87X)      | 0.051  |
|              | 200/10/4           | 0.129 (0.009X)               | 8.631 (0.548X)              | 657.518 (21.13X)         | 6576.72 (282.79X)      | 0.046  |
|              | 200/10/8           | 0.527 (0.130X)               | 23.313 (5.48X)              | 1582.34 (594.73X)        | 15823.4 (5947.3X)      | 0.032  |
| 1000         | 125/375/2          | 101.946 (0.002X)             | 7746.76 (1104.355X)         | ∞                          | ∞                      | 1.887  |
|              | 2000/10/5          | 1181.64 (0.110X)             | 877.197 (24.624X)           | 1582.34 (594.73X)        | 15823.4 (5947.3X)      | 0.046  |
|              | 8000/400/3         | 5867.95 (0.832X)             | ∞                          | ∞                          | ∞                      | 310.10 |

Fig. 4. $2^{100}$-state chain dynamics prediction of a homogeneous, discrete-time Markov chain by the BESTFIT-Extension method with varying maximum node in-degree $k$ and training sets. (a) 100/10/2. (b) 200/10/4. (c) 200/10/8.

Fig. 5. $2^{1000}$-state chain dynamics prediction of a homogeneous, discrete-time Markov chain by the BESTFIT-Extension method with varying maximum node in-degree $k$ and training sets.

multistep transition probabilities of the SCNFN. In all cases of network sizes $N = 10, 100, 1000$, we inferred 100-step transition probabilities (parameters) $P^{ij}(\lambda_j, \tau)$, for $\lambda_j \in \{0, 1\}^N, i = 1, 2, \ldots, N, \tau = 100$. Note that this also implies the inference of the $2^N, \tau$-step transition probabilities of moving from state $\lambda_j$ to state $\mu \in \{0, 1\}^N$, i.e., $P^{(i)}(\mu|\lambda_j, \tau)$, where

$$P(\mu|\lambda_j, \tau) \triangleq \text{Probability}\{\Psi^j(\lambda_j) = \mu\} \quad (43)$$

since it holds that

$$P(\mu|\lambda_j, \tau) = \prod_{i=1}^{N} P^{(i)}(\lambda_j, \mu) \mu^{(i)}(1 - P^{(i)}(\lambda_j, \tau))^{1 - \mu^{(i)}}. \quad (44)$$

More specifically, the convergence of the transition probabilities at a node level in (39) is guaranteed by the central limit theorem; that is, the estimation will converge to the normal distribution $\mathcal{N}(P^{(i)}(\lambda_j, \tau), P^{(i)}(\lambda_j, \tau)(1 - P^{(i)}(\lambda_j, \tau)))$ for large $R$. Eventually, the convergence of the transition probability at the system level in (44) stems from the multivariate delta method [43]. The initial system states $\lambda_j$ were uniformly selected out of the $2^N$ possible logical values. The results reported are averaged across $I = 10^6$ iterations (different cases of initial states $\lambda_j$). $R = 5000$ samples were drawn so as to approximate the transition probabilities. We tested the convergence of the parameters for individual nodes $P^{(i)}(\lambda_j, \tau)$. As a convergence diagnostic metric, we use the mean autocorrelation coefficient

$$\rho(\text{lag}) = \frac{\sum_{j=1}^{I} \rho_j(\text{lag})}{I}$$

B. Transition Probability Estimation

We now evaluate the computational cost and convergence of the sampling scheme described above (39) in order to estimate...
simulate the model for larger However, due to limited computational resources, we could not which exhibited the same trend as shown in Fig. 6(a) and (b).

where
\[ \rho_j(lag) = \frac{\sum_{i=1}^{N} \rho_j^{(i)}(lag)}{N} \]  
(45)

for \( lag = 1, 2, \ldots, R \). \( \rho_j^{(i)}(lag) \) is the autocorrelation coefficient for the parameter \( P_l^{(i)}(l_j, \tau) \) of the individual node \( x_i \), and it is defined as
\[
\rho_j^{(i)}(lag) = \frac{\sum_{m=1}^{R-lag} (\tilde{p}_j^{(i)}(m) - \bar{\tilde{p}}_j^{(i)})(\tilde{p}_j^{(i)}(m+lag) - \bar{\tilde{p}}_j^{(i)})}{\sum_{m=1}^{R-lag} (\tilde{p}_j^{(i)}(m) - \bar{\tilde{p}}_j^{(i)})^2} \]
(46)

where
\[
\tilde{p}_j^{(i)}(m) = \sum_{m=1}^{m} \tilde{x}_j(x_i | l_j), \quad \bar{\tilde{p}}_j^{(i)} = \frac{\sum_{m=1}^{R} \tilde{p}_j^{(i)}(m)}{R}. \]
(47)

In (47), \( lag = 0 \), implies \( \rho_j^{(i)}(lag) = 1 \). The number of samples \( lag \) for the inference is sufficiently large, once the estimations \( \tilde{p}_j^{(i)}(m) \) and \( \tilde{p}_j^{(i)}(m + lag) \) become almost uncorrelated. \( \rho_j^{(i)}(lag) \) is close to zero.) Finally, we average \( \rho_j(lag) \) across all \( I \) initial states [see (45)]. In all cases of network sizes, the autocorrelation coefficient approaches zero after roughly 2000 samples and stabilizes there beyond this number [see Fig. 6(a)]. Furthermore, according to Fig. 6(b), the inference time grows linearly with respect to the number of nodes in the network. At this point, we would like to point out that we obtained results for \( \tau = 1000 \) and \( I = 10^3 \) which exhibited the same trend as shown in Fig. 6(a) and (b). However, due to limited computational resources, we could not simulate the model for larger \( I \); therefore, we omitted these results. Finally, it is worthwhile mentioning that learning the transition probability matrix of the network dynamics as an SCNFN renders it amenable to hardware-accelerated simulation and as a consequence, to a potential 3000× speedup [51]. Furthermore, it could facilitate subsequent domain-specific analysis such as genetic engineering and control [40].

VIII. EXPERIMENTAL RESULTS—REAL WORLD NETWORKS

In this section, we employ the proposed model in order to reverse engineer the nonlinear dynamics of complex biomolecular networks. In particular, we will focus on cell signaling networks. Cell signaling networks describe the time course of the activity of biological features such as proteins, ribonucleic acids (RNAs), and metabolites, and their network-wide causal relationships at the proteomic level. Dynamic modeling of cellular systems is crucial to elucidating important mechanisms such as proliferation, differentiation, and apoptosis [52], [53]. Moreover, accurate understanding of their dynamics paves the way to design synthetic circuits with a desired signaling behavior [40].

In this paper, very similar to the work in [54], we rely on already identified ordinary differential equations (ODEs) in order to demonstrate the agreement of the dynamics generated by the logic model and that by the nonlinear systems. The ODEs describe the biochemical reactions of the models for our case studies, and they were assembled from experimental data and domain knowledge. Berestovsky and Nakhleh [54] utilize the BESTFIT method, whose limitations have already been highlighted in Section VII along with other computationally similar methods. Moreover, BESTFIT is currently broadly used by the computational biology community [46] and supported by the modern tools [47].

A. Case Studies: The Extrinsic Apoptosis Reaction Model and the RAS Executable Model

We use two examples for modeling protein–protein interactions within a cell from cancer biology as test cases; the extrinsic apoptosis reaction model (EARM) [52] and the RAS executable model (REM) [53]. The EARM v1.0 (“EARM”) describes the dynamics of death in single cells (apoptosis) and has been validated in single-cell studies using mouse drugs, pathway-wide RNAi, and protein overexpression. It is a mass-action model based on networks of ODEs which describe proteins known to regulate potent effector caspases (caspases-3 and caspases-7), and which are indicators of apoptosis since they are activated by events such as deoxyribonucleic acid damage (intrinsic cell death pathway) and by stimuli such as tumor necrosis factor (TNF) and TNF-related-apoptosis-inducing ligand (extrinsic cell death pathway). EARM contains 58 features. On average, a feature interacts with three other features in the EARM, and at most with nine.

The second dynamic network comes from the REM which models the signaling pathways of the RAS family proteins (K-RAS, N-RAS, and H-RAS), including their regulators and effectors, at a biochemical level of detail. The RAS oncogene family has been very extensively studied over the last several decades, and the scientific interest is justified since RAS proteins are fundamentally involved in pathological processes such as cancer and in cell physiological processes controlling cellular proliferation and growth. A short version of the REM was considered in this paper, which consists of 438 features. On average, a feature interacts with 23 other features in the REM, and at most with 245.

B. Methodology

We have simulated the models by using PySB (Python framework for systems’ biology modeling) [55] in order to numerically solve the ODEs. The initial states were generated...
the models. The validation data (used for testing) consisted of
features when the SCNFN is trained with only 10 temporal
sequences. Training with more sequences did not achieve significantly
improvement and are left for future research. The reader
temoral delays or sparsity enforcement (see Section IX) could
emerges after training with 1000 time series for the REM.

As corroborated by Fig. 7, the dynamics of the EARM can
be recovered with roughly 15% wrongly predicted states of
features when the SCNFN is trained with only 10 temporal
sequences of a single transition. The highest predictable capac-
ity is achieved by training with 1000 time series. According
to Fig. 7, predictability (36% < 50% misprediction ratio)
emerges after training with 1000 time series for the REM.
Training with more sequences did not achieve significantly
better accuracy. Model extensions such as the incorporation of
temporal delays or sparsity enforcement (see Section IX) could
yield improvement and are left for future research. The reader
may find the computation time required by SCNFLearn in the
Supplementary Material. In [54], results on data coming from
a four-ODE toy system are reported. Moreover, in [54], only
one-step prediction results are reported assuming that the pre-
vious state of the system is known. In contrast, SCNFLearn
succeeds in predicting long-run dynamics when only the initial
states are considered known.

C. Discussion

As corroborated by Fig. 7, the dynamics of the EARM can
be recovered with roughly 15% wrongly predicted states of
features when the SCNFN is trained with only 10 temporal
sequences of a single transition. The highest predictable capac-
ity is achieved by training with 1000 time series. According
to Fig. 7, predictability (36% < 50% misprediction ratio)
emerges after training with 1000 time series for the REM.
Training with more sequences did not achieve significantly
better accuracy. Model extensions such as the incorporation of
temporal delays or sparsity enforcement (see Section IX) could
yield improvement and are left for future research. The reader
may find the computation time required by SCNFLearn in the
Supplementary Material. In [54], results on data coming from
a four-ODE toy system are reported. Moreover, in [54], only
one-step prediction results are reported assuming that the pre-
vious state of the system is known. In contrast, SCNFLearn
succeeds in predicting long-run dynamics when only the initial
states are considered known.

IX. FUTURE RESEARCH

The learning algorithm can be inherently extended to
incorporate prior knowledge of the network structure. This
can be accomplished by modifying the set of the avail-
able literals $L$ so that it only includes literals related to
nodes with which it is known that a node interacts (hard
structural constraints) or by modifying the score function in
the CNFDisjunctionLearn method so that it prioritizes
literals related to these nodes (soft structural constraints).

A temporal Boolean network [48], in which the next state
of the system can depend on many states in the past, can
also be learned by augmenting the set in the right-hand
side of (8) and by extension the set $L$ so that it includes
pairs of literals (for the variables or their negation) for more
than one past states of a node. The set of the available
literals $L$ can also be augmented in order to include dis-
turbance or control inputs. Furthermore, the score function
used in CNFDisjunctionLearn (Line 20, Algorithm 3) can
be modified in order to promote sparsity, i.e., literals
already used are preferred. SCNFN can also be extended to
a context-sensitive SCNFN (the switching time to another BN
is randomly selected) by replacing $a_j^{(i)}$ with $\gamma(t) \land a_j^{(i)}(t) \lor
\neg \gamma(t) \land a_j^{(i)}(t - 1)$, where $\gamma(t)$ is a Bernoulli random variable
common to all nodes and the switching probability as a
parameter, and $a_j^{(i)}(t)(a_j^{(i)}(t - 1))$ the outcome of the $j$th
random literal of node $x_i$ at time $t(t - 1)$. Time-dependent
activation probabilities can also be considered. Learning algo-
rithms which account for these model modifications are left for
future research. Computational learning theory results similar
to those in [36] and [35] for the PBN could also be of interest.
Toward this end, consideration of $k$-CNF formulas instead of
general CNFs could simplify a sample complexity analysis.

Moreover, the definition of the essential variables for node
$x_i$ as defined in [45] for BNs has to be modified in the context
of PBNs and the influence graph has to be weighted. One
suitable metric for the influence (weight) of node $x_i$ on node
$x_j$ can be the sum of the $2^{-N-1}$ Kullback–Leibler divergences
of the two Bernoulli distributions of node $x_j$ when node $x_i$
set to 0 and 1 across all possible states for the rest of nodes
$x_1, \ldots, x_{j-1}, x_{j+1}, \ldots, x_N$. Naive computation requires the
traversal of the transition probability matrix. Development of
efficient statistical methods which use the structure of the
SCNFN in order to estimate the statistical influence of a node
in large networks are left for future research. These methods
could potentially be based on the random sampling of the state
space and the formation of hypothesis tests. We should note
that the SCNFN in its current form is not simplified. Model
reduction techniques which eliminate statistically insignificant
variables could be also important. The development of statis-
tical propositional satisfiability problem solvers for Boolean
satisfiability (i.e., whether a clause will be satisfied with at
least some probability) is inherently based on the stochastic
conjunctive normal form (SCNF). Finally, SCNF clauses can
model stochasticity in hardware designs and lead to advances
in statistical model checking.

X. CONCLUSION

We have introduced a novel modeling framework of PBN
, namely, the SCNFN. We proved that both PBNs and SCNFNs
can represent the same class of Boolean relationships between
the nodes of a network. The adoption of PBNs by a wide spec-
trum of scientific domains has stimulated an intense research
effort on network identification in recent years. However,
the current approaches face one or more of the following
limitations.

1) Prior knowledge of the network structure and/or the node
interactions is assumed so that the parametric portion of
the PBN that can critically influence the network
dynamics can be estimated from observed time series data.

2) Only the candidate Boolean rules which regulate the behavior of a node are learned, while the estimation of the corresponding selection probabilities is ignored.

3) Very strong assumptions are made on the structure of the Boolean relationships. In most of the approaches, a maximum number \( k \) of nodes which participate in the Boolean functions of the network is assumed known a priori, for the learning to be computationally manageable by these methods, \( k \) has to be very small, usually 2 or 3 even for middle-sized networks (i.e., 100 nodes). Additional restrictions can pertain to the complexity of the functions, i.e., the nodes can interact only through AND or OR operations.

4) Approaches which attempt to deal with both the parameter and logical parts of the PBN fail to model in a “compressed” way the transition probability matrix of the PBN and rely on the estimation of a large number of parameters. Therefore, they have unrealistic sample and runtime requirements.

5) Current methods do not report prediction accuracy for large (roughly more than 10 node) networks.

The SCNFN is a compact (in terms of the involved parameters) representation of the PBN which enables an efficient learning algorithm with low sample and runtime demands. The learning procedure circumvents all of the above limitations by greedily identifying the node interactions and approximately maximizing the likelihood of the observed temporal sequences. Subsequently, the SCNFN turns to be an efficient modeling structure in the sense that it can be efficiently sampled/simulated in order to obtain long-run transition probabilities of the generated Markov chain.

ACKNOWLEDGMENT

I. Apostolopoulou would like to thank the Alexander S. Onassis Public Benefit Foundation, Athens, Greece, for awarding her a graduate fellowship that funded this work. The work of I. Apostolopoulou was supported in part by the Alexander Onassis Foundation, in part by ATK/Nick G. Vlahakis, in part by the A. G. Leventis Foundation, in part by the Gerondellis Foundation, and in part by the Carnegie Mellon University College of Engineering Dean’s Fellowship. The authors would like to thank A. Platanios, H. Zhao, F. Condessa, P. Xie, D. Bayani, and W. Guss for the helpful discussions and reviews of this paper. They would like to thank B. Gyori for providing us with the PySB models. They would also like to thank the anonymous reviewers for their valuable comments and suggestions to improve the quality of this paper. An extended version of this paper has been uploaded on arXiv.

REFERENCES

[1] S. A. Kauffman, “Metabolic stability and epigenesis in randomly constructed genetic nets,” J. Theor. Biol., vol. 22, no. 5, pp. 437–467, 1969.

[2] S. A. Kauffman, The Origins of Order: Self-Organization and Selection in Evolution, Oxford Univ. Press, 1993.

[3] J. Heidel, J. Maloney, C. Farrow, and J. A. Rogers, “Finding cycles in synchronous Boolean networks with applications to biochemical systems,” Int. J. Bifurcation Chaos, vol. 13, no. 3, pp. 535–552, 2003.

[4] L. P. Wang, E. E. Pichler, and J. Ross, “Oscillations and chaos in neural networks: An exactly solvable model,” Proc. Nat. Acad. Sci. USA, vol. 87, no. 23, pp. 9467–9471, 1990.

[5] K. E. Kurten, “Correspondence between neural threshold networks and Kauffman Boolean cellular automata,” J. Phys. A, Math. Gen., vol. 21, no. 11, p. L615, 1988.

[6] A. Rolli, M. Manfroini, C. Pincioli, and M. Birattari, “On the design of Boolean network robots,” in Applications of Evolutionary Computation, 2011, pp. 43–52.

[7] M. A. L. Caetano and T. Yoneyama, “Boolean network representation of contagion dynamics during a financial crisis,” Phys. A, Stat. Mech. Appl., vol. 417, pp. 1–6, Jan. 2015.

[8] I. Shmulevich, E. R. Dougherty, S. Kim, and W. Zhang, “Probabilistic Boolean networks: A rule-based uncertainty model for gene regulatory networks,” Bioinformatics, vol. 18, no. 2, pp. 261–274, 2002.

[9] I. Shmulevich, E. R. Dougherty, and W. Zhang, “From Boolean to probabilistic Boolean networks as models of genetic regulatory networks,” Proc. IEEE, vol. 99, no. 11, pp. 1778–1792, Nov. 2002.

[10] P. Trairatphisan, A. Mizera, J. Pang, A. A. Tantar, J. Schneider, and T. Sauter, “Recent development and biomedical applications of probabilistic Boolean networks,” Cell Commun. Signaling, vol. 11, no. 1, p. 46, 2013.

[11] R. Liang, Y. Qiu, and W. K. Ching, “Construction of probabilistic Boolean network for credit default data,” in Proc. 7th Int. Joint Conf. Comput. Sci. Opt. (CSCO), Jul. 2014, pp. 11–15.

[12] S. Marshall, L. Yu, Y. Xiao, and E. R. Dougherty, “Inference of a probabilistic Boolean network from a single observed temporal sequence,” EURASIP J. Bioinform. Syst. Biol., vol. 2007, p. 5, Jan. 2007.

[13] W.-K. Ching, M. M. Ng, E. S. Fung, and T. Akutsu, “On construction of stochastic genetic networks based on gene expression sequences,” Int. J. Neural Syst., vol. 15, no. 4, pp. 297–310, 2005.

[14] W.-K. Ching, X. Chen, and N.-K. Tsing, “Generating probabilistic Boolean networks from a prescribed transition probability matrix,” IET Syst. Biol., vol. 3, no. 6, pp. 453–464, Nov. 2009.

[15] H. Láhdesmäki, S. Hautanen, M. I. Shmulevich, and O. Yli-Harja, “Relationships between probabilistic Boolean networks and dynamic Bayesian networks as models of gene regulatory networks,” Signal Process., vol. 86, no. 4, pp. 814–834, 2006.

[16] T. W. Anderson and L. A. Goodman, “Statistical inference about Markov chains,” Ann. Math. Statist., vol. 28, no. 1, pp. 89–110, 1957.

[17] J. Zhong, J. Lu, T. Huang, and D. W. C. Ho, “Controllability and synchronization analysis of identical-hierarchy mixed-valued logical control networks,” IEEE Trans. Cybern., vol. 47, no. 11, pp. 3482–3493, Nov. 2017.

[18] M. Meng, J. Lam, J.-E. Feng, and K. C. Chen, “Stability and guaranteed cost analysis of time-triggered Boolean networks,” IEEE Trans. Neural Netw. Learn. Syst., vol. 29, no. 8, pp. 3893–3899, Aug. 2018.

[19] Y. Guo, “Observability of Boolean control networks using parallel extension and set reachability,” IEEE Trans. Neural Netw. Learn. Syst., vol. 29, no. 12, pp. 6402–6408, Dec. 2018.

[20] Y. Wu and T. Shen, “Policy iteration algorithm for optimal control of stochastic logical dynamical systems,” IEEE Trans. Neural Netw. Learn. Syst., vol. 29, no. 5, pp. 2031–2036, May 2018.

[21] Y. Zhao, B. K. Ghosh, and D. Cheng, “Control of large-scale Boolean networks via network aggregation,” IEEE Trans. Neural Netw. Learn. Syst., vol. 27, no. 7, pp. 1527–1536, Jul. 2016.

[22] R. Liu, J. Lu, Y. Liu, J. Cao, and Z.-G. Wu, “Delayed feedback control for stabilization of Boolean control networks with state delay,” IEEE Trans. Neural Netw. Learn. Syst., vol. 29, no. 7, pp. 3283–3288, Jul. 2018.

[23] Y. Liu, J. Cao, B. Li, and J. Lu, “Normalizability and solvability of dynamic-algebraic Boolean networks,” IEEE Trans. Neural Netw. Learn. Syst., vol. 30, no. 9, pp. 3301–3306, Jul. 2018.

[24] F. Li, H. Yan, and H. R. Karimi, “Single-input pinning controller design for reachability of Boolean networks,” IEEE Trans. Neural Netw. Learn. Syst., vol. 29, no. 7, pp. 3264–3269, Jul. 2018.

[25] F. Li, “Pinning control design for the stabilization of Boolean networks,” IEEE Trans. Neural Netw. Learn. Syst., vol. 27, no. 7, pp. 1585–1590, Jul. 2016.

[26] L. Zhang and K. Zhang, “Controllability and observability of Boolean control networks with time-variant delays in states,” IEEE Trans. Neural Netw. Learn. Syst., vol. 24, no. 9, pp. 1478–1484, Sep. 2013.

[27] D. Cheng and H. Qi, “Controllability and observability of Boolean control networks,” Automatica, vol. 45, no. 7, pp. 1659–1667, 2009.
A. Mizera, J. Pang, C. Su, and Q. Yuan, “ASSA-PBN: A toolbox for probabilistic Boolean networks via the semi-tensor product method,” IEEE Trans. Neural Netw. Learn. Syst., vol. 24, no. 6, pp. 996–1001, Jun. 2013.

H. Zhang, H. Tian, Z. Wang, and Y. Hou, “Synchronization analysis and design of coupled Boolean networks based on periodic switching sequences,” IEEE Trans. Neural Netw. Learn. Syst., vol. 27, no. 12, pp. 2754–2759, Dec. 2016.

J. Zhong, J. Lu, Y. Liu, and J. Cao, “Synchronization in an array of output-coupled Boolean networks with time delay,” IEEE Trans. Neural Netw. Learn. Syst., vol. 25, no. 12, pp. 2288–2294, Dec. 2014.

D. Cheng and H. Qi, “A linear representation of dynamics of Boolean networks,” IEEE Trans. Autom. Control, vol. 55, no. 10, pp. 2251–2258, Oct. 2010.

L. Tong, Y. Liu, J. Lou, J. Lu, and F. E. Alsaadi, “Static output feedback set stabilization for context-sensitive probabilistic Boolean control networks,” Appl. Math. Comput., vol. 332, pp. 263–275, Sep. 2018.

A. A. Melkman, X. Cheng, W.-K. Ching, and T. Akutsu, “Identifying a probabilistic Boolean threshold network from samples,” IEEE Trans. Neural Netw. Learn. Syst., vol. 29, no. 4, pp. 869–881, Apr. 2018.

X. Cheng, T. Mori, Y. Qiu, W.-K. Ching, and T. Akutsu, “Exact identification of the structure of a probabilistic Boolean network from samples,” IEEE/ACM Trans. Comput. Biol. Bioinf., vol. 13, no. 6, pp. 1107–1116, Nov. 2016.

X. Zhang, H. Han, and W. Zhang, “Identification of Boolean networks using preimuned network topology information,” IEEE Trans. Neural Netw. Learn. Syst., vol. 29, no. 2, pp. 464–469, Feb. 2017.

K. Kobayashi and K. Hiraishi, “Design of probabilistic Boolean networks based on network structure and steady-state probabilities,” IEEE Trans. Neural Netw. Learn. Syst., vol. 28, no. 8, pp. 1966–1971, Aug. 2017.

M. Choi, J. Shi, S. H. Jung, X. Chen, and K.-H. Cho, “Attractor landscape analysis reveals feedback loops in the p53 network that control the cellular response to DNA damage,” Sci. Signaling, vol. 5, no. 251, pp. ra83–ra83, 2012.

A. A. Nielsen et al., “Genetic circuit design automation,” Science, vol. 352, no. 6281, p. aaq7341, 2016.

V. Pahuja, R. K. Layek, and P. Mitra, “Learning a probabilistic Boolean network model from biological pathways and time-series expression data,” in Proc. 38th Annu. Int. Conf. IEEE Eng. Med. Biol. Soc. (EMBC), Aug. 2016, pp. 1471–1475.

A. Mizera, J. Pang, C. Su, and Q. Yuan, “ASSA-PBN: A toolbox for probabilistic Boolean networks,” IEEE/ACM Trans. Comput. Biol. Bioinf., vol. 15, no. 4, pp. 1203–1216, Jul/Aug. 2017.

L. A. Wasserman, All of Statistics: A Concise Course in Statistical Inference, Springer, 2013.

E. A. Plataniotis, A. Blum, and T. M. Mitchell, “Estimating accuracy from unlabeled data,” in Proc. UAI, 2014, pp. 682–691.

I. Shmulevich, A. Saarinen, O. Yli-Harja, and J. Astola, “Inference of genetic regulatory networks via best-fit extensions,” in Computational and Statistical Approaches to Genomics. Boston, MA, USA: Springer, 2003, pp. 197–210.

S. N. Steinway, M. B. Biggs, T. P. Loughran, J. A. Papin, and R. Albert, “Inference of network dynamics and metabolic interactions in the gut microbiome,” PLoS Comput. Biol., vol. 11, no. 6, p. e1004338, 2015.

C. Müssel, M. Hoffensitz, and H. A. Kestler, “BoolNet—An R package for generation, reconstruction and analysis of Boolean networks,” Bioinformatics, vol. 26, no. 10, pp. 1378–1380, 2010.

A. Silvescu and V. Honavar, “Temporal Boolean network models of genetic networks and their inference from gene expression time series,” Complex Syst., vol. 13, no. 1, pp. 61–78, 2001.

T. M. Mitchell et al., Machine learning. WCB, Boston, MA, USA: McGraw-Hill, 1997.

A. Fauré, A. Naldi, C. Chaouiya, and D. Thieffry, “Dynamical analysis of a generic Boolean model for the control of the mammalian cell cycle,” Bioinformatics, vol. 22, no. 14, pp. e124–e131, 2006.

N. Miskov-Zivanov et al., “Regulatory network analysis acceleration with reconﬁgurable hardware,” in Proc. EMBC, Aug./Sep. 2011, pp. 149–152.

J. G. Albeck, J. M. Burke, S. L. Spencer, D. A. Lauffenburger, and P. K. Sorger, “Modeling a snap-action, variable-delay switch controlling extrinsc cell death,” PLoS Biol., vol. 6, no. 12, p. e299, 2008.

P. Sorger and D. Pratt. RAS Executable Model (REM). (2016). [Online]. Available: http://rasmodel.org/index.html

N. Berestovsky and L. Kakhle, “An evaluation of methods for inferring Boolean networks from time-series data,” PLoS ONE, vol. 8, no. 6, p. e66031, 2013.

C. F. Lopez, J. L. Muhlich, J. A. Bachman, and P. K. Sorger, “Programming biological models in Python using PySB,” Mol. Syst. Biol., vol. 9, no. 1, p. 646, 2013.

Ifigenia Apostolopoulou (S’18) received the B.S. degree in electrical and computer engineering from the University of Thessaly, Volos, Greece, in 2013. She is currently pursuing the Ph.D. degree in machine learning with Carnegie Mellon University, Pittsburgh, PA, USA.

In 2015, she joined the Electrical and Computer Engineering Department, Carnegie Mellon University, as a Ph.D. Student.

Diana Marculescu (S’94–M’98–SM’09–F’15) received the Dipl.Ing. degree in computer science from the Politehnica University of Bucharest, Bucharest, Romania, in 1991, and the Ph.D. degree in computer engineering from the University of Southern California, Los Angeles, CA, USA, in 1998.

She is currently with the Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA, USA, where she is a Professor of electrical and computer engineering.

Dr. Marculescu was selected as an ELATE Fellow from 2013 to 2014. She was a recipient of the National Science Foundation Faculty Career Award from 2000 to 2004, the ACM SIGDA Technical Leadership Award in 2003, the Carnegie Institute of Technology George Tallman Ladd Research Award in 2004, the Australian Research Council Future Fellowship from 2013 to 2017, the Marie R. Pistilli Women in EDABest Paper Award in 2014, and several best paper awards. She was a Distinguished Lecturer of the IEEE Circuits and Systems Society from 2004 to 2005 and the Chair of the Association for Computing Machinery Special Interest Group on Design Automation from 2005 to 2009. She chaired several conferences and symposia in her area. She is currently an Associate Editor of the IEEE TRANSACTIONS ON COMPUTERS. She is a Distinguished Scientist of ACM.