Proceedings of the 18th International Conference on Principles of Knowledge Representation and Reasoning
Main Track

Boolean Network Learning in Vector Spaces for Genome-wide Network Analysis

Taisuke Sato [1] , **Ryosuke Kojima**[2]
[1]National Institute of Informatics (NII), Tokyo, Japan
[2]Kyoto University, Kyoto, Japan
satou_taisuke@nii.ac.jp, kojima.ryosuke.8e@kyoto-u.ac.jp

Abstract

Boolean networks (BNs) are one of the standard tools for modeling gene regulatory networks in biology but their learning has been limited to small networks due to computational difficulty. Aiming at unprecedented scalability, we focus on a subclass of BNs called AND/OR Boolean networks where Boolean formulas are restricted to a conjunction or a disjunction of literals. We represent an AND/OR BN with N nodes by an $N \times 2N$ binary matrix $\boldsymbol{Q}$ paired with an N dimensional integer vector $\boldsymbol{\theta}$ called a threshold vector, a state of the BN by an N dimensional binary state vector $\boldsymbol{s}$ and a state transition by matrix operations on $\boldsymbol{Q}, \boldsymbol{\theta}$ and $\boldsymbol{s}$. Given a list of state transitions $\boldsymbol{S} = [\boldsymbol{s}_0 \cdots \boldsymbol{s}_L]$, we learn $\boldsymbol{Q}$ and $\boldsymbol{\theta}$ in a continuous space by minimizing a cost function $J(\tilde{\boldsymbol{Q}}, \boldsymbol{\theta}, \boldsymbol{S})$ w.r.t. a real number matrix $\tilde{\boldsymbol{Q}}$ and $\boldsymbol{\theta}$ while thresholding $\tilde{\boldsymbol{Q}}$ into a binary matrix $\boldsymbol{Q}$ using $\boldsymbol{\theta}$ so that $\boldsymbol{Q}$ represents an AND/OR BN realizing the target state transitions $\boldsymbol{S}$. We conducted experiments with artificial and real data sets to check scalability and accuracy of our learning algorithm. First we randomly generated AND/OR BNs up to N=5,000 nodes and empirically confirmed $O(N^2)$ learning time behavior using them. We also observed 99.8% bit-by-bit prediction accuracy[1] with state transition data generated by AND/OR BNs. For real data, we learned genome-wide AND/OR BNs with $10,928$ nodes for budding yeast from transcription profiling data sets, each containing $10,928$ mRNAs and 40 transitions and achieved for instance 84.3% prediction accuracy and successfully extracted more than 6,000 small AND/ORs whose average prediction accuracy reaches much higher 94.9%.

1 Introduction

*Boolean network*s (BNs) introduced by Kauffman (Kauffman 1993) are one of the standard tools for modeling biological networks and have been primarily used to model gene regulatory networks. A BN is a finite discrete state transition system described by a directed graph where nodes typically represent genes. When there are N nodes, their state is collectively represented by an N dimensional column vector $[x_1 \ \ldots \ x_N]^T$ called *state vector* such that $x_i \in \{1(\mathrm{true}), 0(\mathrm{false})\}$ denotes the state of node i ($1 \leq i \leq N$). The state transition is specified by a Boolean function f_i associated with node i like $x_i(t+1) = f_i(x_{k_1}(t), \ldots, x_{k_m}(t))$ where $\{k_1, \ldots, k_m\}$ is the set of incoming neighbors of node i whose state at time t is $[x_{k_1}(t) \ \ldots \ x_{k_m}(t)]^T$. In this paper we only deal with non-probabilistic synchronous BNs where state transitions occur simultaneously.

Although it is usual to represent Boolean functions by Boolean formulas, it is also possible to represent them by matrices and vectors. For example, Cheng and Qi proposed matricized representation of Boolean functions where truth values are represented by $[1\,0]^T$(true) and $[0\,1]^T$(false), and a Boolean function $f(x_1, \ldots, x_N)$ by $\boldsymbol{F}(\boldsymbol{x}_1 \ltimes \cdots \ltimes \boldsymbol{x}_N)$ using a semi-tensor product $\ltimes$[2] where $\boldsymbol{F}$ is a 2×2^N binary matrix and $\boldsymbol{x}_i \in \{[1\,0]^T, [0\,1]^T\}$ ($1 \leq i \leq N$). Their semi-tensor product encoding however yields a $2^N \times 2^N$ matrix to describe the state transition of a BN (Cheng and Qi 2005; Cheng and Qi 2010).

Later Kobayashi and Hiraishi gave a similar treatment of BN expression using Kronecker product and reduced the matrix size from $2^N \times 2^N$ to $2N \times 2^N$ by treating Boolean functions individually (Kobayashi and Hiraishi 2014). Nonetheless, their approach still needs to operate on data of exponential size. Or more generally, as long as arbitrary Boolean functions are allowed, scalable network learning seems hardly possible, be it symbolic or numerical. Such computational difficulty is particularly painful when we challenge the learning of genome-wide BNs and also explains why BN learning remains on a relatively small scale (Akutsu, Miyano, and Kuhara 1999; Lähdesmäki, Shmulevich, and Yli-Harja 2003; Li et al. 2004; Martin et al. 2007; Fogelberg and Palade 2009; Higa, Louzada, and Hashimoto 2010; Inoue, Ribeiro, and Sakama 2014; Xu et al. 2014; Ouyang et al. 2014; Barman and Kwon 2018; Joo et al. 2018; Chevalier et al. 2019; Wilson et al. 2019).

In this paper, we propose a novel approach to the learning of large scale BNs. First we restrict BNs to simplified ones called AND/OR BNs where there are only two types of Boolean formula; one is a conjunction and the other a disjunction of literals. Second we express an AND/OR BN containing N nodes as a $N \times 2N$ binary matrix $\boldsymbol{Q}$ paired with an N dimensional integer vector $\boldsymbol{\theta}$ called *threshold vector*. Third we learn $\boldsymbol{Q}$ and $\boldsymbol{\theta}$ from state transition data

[1]We use "prediction accuracy" as accuracy for unseen data. In the case of cross validation, prediction accuracy = 1 - test error.

[2]Semi-tensor product is a generalization of matrix product which preserves many properties of matrix production such as the laws of distribution and association.

in a vector space instead of searching for Boolean functions in a symbolic space (Inoue, Ribeiro, and Sakama 2014; Chevalier et al. 2019) or in a discrete space (Lähdesmäki, Shmulevich, and Yli-Harja 2003; Li et al. 2004; Martin et al. 2007; Higa, Louzada, and Hashimoto 2010; Xu et al. 2014; Ouyang et al. 2014; Barman and Kwon 2018). We minimize a cost function $J(\tilde{\boldsymbol{Q}}, \boldsymbol{\theta})$ w.r.t. a real number matrix $\tilde{\boldsymbol{Q}}$ and $\boldsymbol{\theta}$ while thresholding $\tilde{\boldsymbol{Q}}$ to a binary matrix $\boldsymbol{Q}$ by $\boldsymbol{\theta}$ to obtain $\boldsymbol{Q}$ representing an AND/OR BN that best approximates the target state transitions. Although our approach is not guaranteed to find an exact BN, we can expect robustness, computational efficiency and scalability of learning supported by hardware technology such as many-core processors and GPUs.

We conducted learning experiments with artificial data and real data. For artificial data, we empirically observed $O(N^2)$ learning time behavior up to N=5,000 nodes w.r.t. learning data sampled from randomly generated AND/OR BNs. We also observed more than 99.5% prediction accuracy by 10-fold CV with state transition data generated by AND/OR BNs consisting of 100 nodes. Concerning real data, we successfully learned *genome-wide* AND/OR BNs for budding yeast (Saccharomyces cerevisiae) with $10,928$ nodes from transcription profiling data for $10,928$ mRNAs and 40 transitions and achieved for instance 84.3% prediction accuracy and successfully extracted more than 6,000 small AND/ORs whose average prediction accuracy reaches much higher 94.9%.

Our technical contributions include a formulation of AND/OR BN learning problem as cost minimization in vector spaces with a novel cost function, a proposal of a matricized AND/OR BN learning algorithm that works in quadratic time theoretically and empirically and the achievement of full genome-wide AND/OR BN learning for real genome data sets.

2 Preliminaries

Throughout this paper, N stands for the number of Boolean nodes (variables) in a BN, L for the number of observed data (state vectors), bold italic capital letters such as $\boldsymbol{A}$ for a matrix and bold italic lower case letters such as $\boldsymbol{a}$ for a vector. Vectors are treated as a special type of matrix depending on the context. $\mathbf{1}$ specifically denotes an all-one column vector whereas $\mathbb{1}$ denotes an all-one matrix. For a vector $\boldsymbol{a}$, $\boldsymbol{a}(j)$ denotes the j-th component of $\boldsymbol{a}$ and $\boldsymbol{A}(i,j)$ stands for the $(i\,j)$ component of a matrix $\boldsymbol{A}$. $\|\boldsymbol{A}\|_F^2$ is the Frobenius norm of $\boldsymbol{A}$ and $\|\boldsymbol{a}\|_1$ is the 1-norm of $\boldsymbol{a}$.[3] Hereafter we assume in every expression, dimensions of vectors and matrices are always compatible. Let $\boldsymbol{A}$, $\boldsymbol{B}$ be matrices. $\boldsymbol{A}(i,:)$ designates the i-th row vector of $\boldsymbol{A}$ whereas $\boldsymbol{A}(:,j)$ designates the j-th column vector of $\boldsymbol{A}$. $[\boldsymbol{A};\boldsymbol{B}]$ is a matrix formed by vertically concatenating $\boldsymbol{A}$ to $\boldsymbol{B}$. $\boldsymbol{A} \odot \boldsymbol{B}$ is the element-wise product of $\boldsymbol{A}$ and $\boldsymbol{B}$. We now introduce a thresholding operation. For two numbers a and b, define $(a)_{\geq b} = \begin{cases} 1 & \text{if } a \geq b \\ 0 & \text{otherwise} \end{cases}$. For a row vector $\boldsymbol{a} = [a_1 \cdots a_L]$, define $(\boldsymbol{a})_{\geq b} = [(a_1)_{\geq b} \cdots (a_L)_{\geq b}]$. $(\boldsymbol{a})_{<b}$ is similarly defined. For a matrix $\boldsymbol{A} = [\boldsymbol{a}_1; \cdots; \boldsymbol{a}_N]$ comprised of N rows and a column vector $\boldsymbol{b} = [b_1 \cdots b_N]^T$, define $(\boldsymbol{A})_{\geq \boldsymbol{b}} = [(\boldsymbol{a}_1)_{\geq \boldsymbol{b}(1)}; \cdots; (\boldsymbol{a}_N)_{\geq \boldsymbol{b}(N)}]$. As a special case, when $\boldsymbol{a}$ is a column vector $[a_1 \cdots a_L]^T$, $\boldsymbol{a}_{\geq \boldsymbol{b}}$ denotes the component-wise comparison of $\boldsymbol{a}$ and $\boldsymbol{b}$, i.e. $\boldsymbol{a}_{\geq \boldsymbol{b}} = [(a_1)_{\geq \boldsymbol{b}(1)} \cdots (a_L)_{\geq \boldsymbol{b}(L)}]^T$. For a binary matrix $\boldsymbol{A}$, $|\boldsymbol{A}|$ designates the number of 1's in $\boldsymbol{A}$. We use a non-linear function $\min_1(x)$ defined by $\min_1(x) = \min(x, 1)$. When applied to a vector, it is applied element-wise. A literal is a Boolean variable or its negation. The former is called a positive literal whereas the latter is called a negative literal.

[3] $\|\boldsymbol{A}\|_F^2 = \sum_{ij} \boldsymbol{A}(i,j)^2$ and $\|\boldsymbol{a}\|_1 = \sum_i |\boldsymbol{a}(i)|$.

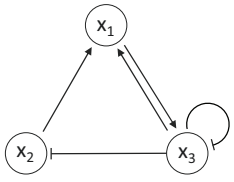
3 Matrix Representation of AND/OR BN

As stated before, a Boolean network (BN) is a directed graph where each node i has a binary state $x_i \in \{1, 0\}$[4] and also has an associated Boolean function $f_i(x_{k_1}, \ldots, x_{k_m})$ that determines the state transition of node i by $x_i(t+1) = f_i(x_{k_1}(t), \ldots, x_{k_m}(t))$ where $x_{k_p}(t)$ $(1 \leq p \leq m)$ denotes the state of node k_p at time t. Correspondingly to $f_i(x_{k_1}, \ldots, x_{k_m})$, in the graph, there is a sharp (resp. blunt) arrow from node $k_p (1 \leq p \leq m)$ to node i when x_{k_p} positively (resp. negatively) affects x_i. Hereafter for concreteness and simplicity, we do not make a distinction between a Boolean formula and a Boolean function it represents and assume that a Boolean formula is associated with each node.

AND/OR BNs are subclass of BNs where associated Boolean formulas are restricted to a conjunction or disjunction of distinct literals (Melkman, Tamura, and Akutsu 2010).[5] There are two types of node, AND node and OR node. An AND node (resp. OR node) is one such that the associated Boolean formula is a conjunction $x_{k_1}^\circ \wedge \cdots \wedge x_{k_m}^\circ$ (resp. disjunction $x_{k_1}^\circ \vee \cdots \vee x_{k_m}^\circ$). Here $x_{k_p}^\circ$ $(1 \leq p \leq m)$ stands for a literal, i.e. $x_{k_p}^\circ = x_{k_p}$ if $x_{k_p}^\circ$ is a positive literal, else $x_{k_p}^\circ = \neg x_{k_p}$.

Figure 1 shows an example of AND/OR BN BN_0 containing three nodes (left) and their state transitions by Boolean formulas (right).

In what follows, BN always means AND/OR BN. Now we introduce matricized representation of BN. Suppose there is



$$\begin{cases} x_1(t+1) = x_2(t) \wedge x_3(t) \\ x_2(t+1) = \neg x_3(t) \\ x_3(t+1) = x_1(t) \vee \neg x_3(t) \end{cases}$$

Figure 1: An AND/OR Boolean network BN_0

[4] We assume that nodes are numbered from 1 to N when there are N nodes and say "node i" to refer to the node numbered i.

[5] In this paper, we use AND/OR as a synonym for conjunction/disjunction.

$$\boldsymbol{Q} = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad \boldsymbol{\theta} = [2\ 1\ 1\ ]^T$$

Figure 2: $\boldsymbol{Q}$ and $\boldsymbol{\theta}$ representing BN_0

a BN with N nodes. To represent it, we use an $N \times 2N$ binary matrix $\boldsymbol{Q}$ that indicates for each node literal occurrences in a Boolean formula associated with the node but distinguishes positive occurrence from negative occurrence of literal. Suppose node i is an AND node with an associated conjunction $C_i = x^\circ_{k_1} \wedge \cdots \wedge x^\circ_{k_m}$. For j ($1 \le j \le 2N$), set $\boldsymbol{Q}(i,j) = 1$ if $j = k_p$ and $x^\circ_{k_p}$ is a positive literal, or $j = k_p + N$ and $x^\circ_{k_p}$ is a negative literal for some k_p ($1 \le p \le m$). Otherwise set $\boldsymbol{Q}(i,j) = 0$. The case of OR node is treated the same way.

$\boldsymbol{Q}$ is not enough to represent the BN because it has no information about *node type*, AND node or OR node, of a node. So we introduce a *threshold vector* $\boldsymbol{\theta}$ which is an integer column vector. The primary role of $\boldsymbol{\theta}$ is to store a threshold value $\boldsymbol{\theta}(i)$ for a node i which also indicates the node type.

Suppose node i is an AND node and the associated Boolean formula is $C_i = x^\circ_{k_1} \wedge \cdots \wedge x^\circ_{k_m}$. Then put $\boldsymbol{\theta}(i) = m$, i.e. $\boldsymbol{\theta}(i)$ is the number of literals in the conjunction C_i in the case of AND node. We can say that C_i is true iff (if-and-only if) the number of true literals in C_i is equal to or greater than $\boldsymbol{\theta}(i)$. Likewise we put $\boldsymbol{\theta}(i) = 1$ if node i is an OR node. Consequently if $\boldsymbol{\theta}(i) > 1$ holds, node i is an AND node. Otherwise it is an OR node but if it contains only one literal, it is also an AND node.

Figure 2 illustrates $\boldsymbol{Q}$ and $\boldsymbol{\theta}$ representing BN_0.

We next show how to compute state transition using $\boldsymbol{Q}$ and $\boldsymbol{\theta}$. Let $\boldsymbol{s} = [x_1 \cdots x_N]^T$ be a state vector of the BN. We introduce a *dualized state vector* $\boldsymbol{s}^d = [\boldsymbol{s}; \mathbf{1}{-}\boldsymbol{s}]$[6] which holds a state $x_i \in \{1,0\}$ of node i ($1 \le i \le N$) redundantly using two bits, $\boldsymbol{s}^d(i)$ and $\boldsymbol{s}^d(i+N)$ in such a way that $x_i = 1$ iff $\boldsymbol{s}^d(i) = 1$ and $\boldsymbol{s}^d(i+N) = 0$, and $x_i = 0$ iff $\boldsymbol{s}^d(i) = 0$ and $\boldsymbol{s}^d(i{+}N) = 1$. Let $\{x^\circ_{k_1}, \ldots, x^\circ_{k_m}\}$ be the set of literals occurring in a Boolean formula associated with node i. It is easy to see that by definition of $\boldsymbol{s}^d$, $\boldsymbol{Q}(i,:)\boldsymbol{s}^d$, a scalar, is equal to the number of true literals in $\{x^\circ_{k_1}, \ldots, x^\circ_{k_m}\}$ in the state $\boldsymbol{s}$. It follows by construction of $\boldsymbol{\theta}$ that $(\boldsymbol{Q}(i,:)\boldsymbol{s}^d)_{\ge\boldsymbol{\theta}(i)}$ is the next state of node i regardless of i's node type and $(\boldsymbol{Q}\boldsymbol{s}^d)_{\ge\boldsymbol{\theta}}$ is the next state vectors of $\boldsymbol{s}$.

Now we extend $\boldsymbol{s}$ to a matrix. Let $\boldsymbol{S} = [\boldsymbol{s}_1 \cdots \boldsymbol{s}_L]$ be an $N \times L$ matrix containing L state vectors. Consider $\boldsymbol{S}^d = [\boldsymbol{S}; \mathbb{1} - \boldsymbol{S}]$.[7] It is a $2N \times L$ matrix containing L dualized state vectors. Then $(\boldsymbol{Q}\boldsymbol{S}^d)_{\ge\boldsymbol{\theta}} = [(\boldsymbol{s}^d_1)_{\ge\boldsymbol{\theta}} \cdots (\boldsymbol{s}^d_L)_{\ge\boldsymbol{\theta}}]$ is a matrix of the next state vectors for the BN.

[6] $\mathbf{1} - \boldsymbol{s}$ is the bit inversion of $\boldsymbol{s}$.

[7] $\mathbb{1} - \boldsymbol{S}$ is the bit inversion of $\boldsymbol{S}$.

4 Matricized BN Learning

4.1 Row-wise Learning and a Cost Function

We consider BN learning in the following setting. We are given a set $\boldsymbol{S}_{in} = [\boldsymbol{s}^{in}_1 \cdots \boldsymbol{s}^{in}_L]$ of L input state vectors and the set $\boldsymbol{S}_{out} = [\boldsymbol{s}^{out}_1 \cdots \boldsymbol{s}^{out}_L]$ of L corresponding output state vectors where $\boldsymbol{s}^{out}_j$ is the next state of $\boldsymbol{s}^{in}_j (1 \le j \le L)$. Our task is to learn a BN that maps $\boldsymbol{S}_{in}$ to $\boldsymbol{S}_{out}$ by state transition. We tackle this task by learning a binary matrix $\boldsymbol{Q}$ and a threshold vector $\boldsymbol{\theta}$ representing a BN such that $\boldsymbol{S}_{out} = (\boldsymbol{Q}\boldsymbol{S}^d_{in})_{\ge\boldsymbol{\theta}}$ holds where $\boldsymbol{S}^d_{in} = [\boldsymbol{S}_{in}; \mathbb{1} - \boldsymbol{S}_{in}]$.

Since two matrices are equal when two corresponding rows are equal for every row in each matrix, this learning task is reduced to the learning of a row binary vector $\boldsymbol{Q}(i,:)$ that satisfies $\boldsymbol{S}_{out}(i,:) = (\boldsymbol{Q}(i,:)\boldsymbol{S}^d_{in})_{\ge\boldsymbol{\theta}(i)}$ for each node i ($1 \le i \le N$). We learn such vectors by cost minimization in a continuous space considering $\boldsymbol{Q}(i,:)$ as a real-valued vector.

Put $\boldsymbol{a}_i = \boldsymbol{S}_{out}(i,:)$ and $\boldsymbol{b}_i = \boldsymbol{Q}(i,:) (1 \le i \le N)$ respectively. For every node i ($1 \le i \le N$), we have to learn a binary row vector $\boldsymbol{b}_i$ satisfying $\boldsymbol{a}_i = (\boldsymbol{b}_i\boldsymbol{S}^d_{in})_{\ge\boldsymbol{\theta}(i)}$. Introduce two non-negative cost functions, $J^{\mathrm{and}}_i(\boldsymbol{b}_i)$ (1) for AND node and $J^{\mathrm{or}}_i(\boldsymbol{b}_i)$ (2) for OR node. Given a node type for node i, we learn $\boldsymbol{b}_i$ by minimizing $J^{\mathrm{and}}_i(\boldsymbol{b}_i)$ to zero if it is an AND node, else by minimizing $J^{\mathrm{or}}_i(\boldsymbol{b}_i)$ to zero if it is an OR node. Once $\boldsymbol{b}_i$ is learned as the minimizer of those cost functions, $\boldsymbol{\theta}(i)$ is automatically computed from $\boldsymbol{b}_i$ and the node type for i.

$$\begin{aligned} J^{\mathrm{and}}_i(\boldsymbol{b}_i) \;=\; & ((\mathbf{1}{-}\boldsymbol{a}_i) \bullet (\mathbf{1} - \min_1(\boldsymbol{b}_i([\mathbb{1};\mathbb{1}]{-}\boldsymbol{S}^d_{in})))) \\ & +(1/2)(\boldsymbol{a}_i \bullet ((\boldsymbol{b}_i \odot \boldsymbol{b}_i)([\mathbb{1};\mathbb{1}]{-}\boldsymbol{S}^d_{in}))) \\ & +(1/2)\|\boldsymbol{b}_i \odot (\mathbf{1}{-}\boldsymbol{b}_i)\|^2_F \end{aligned} \tag{1}$$

$$\begin{aligned} J^{\mathrm{or}}_i(\boldsymbol{b}_i) \;=\; & (\boldsymbol{a}_i \bullet (\mathbf{1}{-}\min_1(\boldsymbol{b}_i\boldsymbol{S}^d_{in}))) \\ & +(1/2)((\mathbf{1}{-}\boldsymbol{a}_i) \bullet ((\boldsymbol{b}_i \odot \boldsymbol{b}_i)\boldsymbol{S}^d_{in})) \\ & +(1/2)\|\boldsymbol{b}_i \odot (\mathbf{1}{-}\boldsymbol{b}_i)\|^2_F \end{aligned} \tag{2}$$

We can prove

Proposition 1. *Suppose node i is an AND node. Then $J^{\mathrm{and}}_i(\boldsymbol{b}_i) = 0$ if-and-only-if $\boldsymbol{b}_i$ is a binary vector satisfying $\boldsymbol{a}_i = (\boldsymbol{b}_i\boldsymbol{S}_{in})_{\ge\boldsymbol{\theta}(i)}$ where $\boldsymbol{\theta}(i)$ is the number of literals in the conjunction associated with node i.*

Proof. Suppose $J^{\mathrm{and}}_i(\boldsymbol{b}_i) = 0$. First we prove $\boldsymbol{b}_i$ is binary. Since every term in (1) is non-negative, $J^{\mathrm{and}}_i = 0$ implies all terms in (1) are zero, in particular we have $\|\boldsymbol{b}_i \odot (\mathbf{1}{-}\boldsymbol{b}_i)\|^2_F = 0$. So $\boldsymbol{b}_i$ is a binary vector. We next prove $\boldsymbol{a}_i = (\boldsymbol{b}_i\boldsymbol{S}_{in})_{\ge\boldsymbol{\theta}(i)}$. We assume $\boldsymbol{b}_i$ represents a conjunction C_i associated with node i. Take an arbitrary j ($1 \le j \le L$) and put $\delta = (\boldsymbol{b}_i([\mathbb{1};\mathbb{1}]{-}\boldsymbol{S}^d_{in}))(j)$. δ is the number of literals in the conjunction C_i which are false in the state $\boldsymbol{S}_{in}(:,j)$. If $\boldsymbol{a}_i(j) = 0$, then $1 - \min_1(\delta) = 0$ follows from the first term in (1) being zero, which implies $\delta \ge 1$. $\delta \ge 1$ in turn implies C_i contains at least one literal false in $\boldsymbol{S}_{in}(:,j)$. Therefore $\boldsymbol{b}_i\boldsymbol{S}^d_{in}(:,j)$, the number of literals in C_i true in $\boldsymbol{S}_{in}(:,j)$, is less than the number of all literals C_i which is equal to $\boldsymbol{\theta}(i)$ by construction of $\boldsymbol{\theta}(i)$. So $\boldsymbol{b}_i\boldsymbol{S}^d_{in}(:,j) < \boldsymbol{\theta}(i)$

holds. Hence we have $(\boldsymbol{b}_i \boldsymbol{S}^d_{in}(:,j))_{\geq \boldsymbol{\theta}(i)} = 0$. Otherwise suppose $\boldsymbol{a}_i(j) = 1$. Then by the second term in (1) being zero, and by the fact that $\boldsymbol{b}_i \odot \boldsymbol{b}_i = \boldsymbol{b}_i$ holds when $\boldsymbol{b}_i$ is binary, we can similarly conclude C_i is true in the state $\boldsymbol{S}_{in}(:,j)$ and $(\boldsymbol{b}_i \boldsymbol{S}^d_{in}(:,j))_{\geq \boldsymbol{\theta}(i)} = 1$ holds. Consequently $\boldsymbol{a}_i(j) = (\boldsymbol{b}_i \boldsymbol{S}^d_{in}(:,j))_{\geq \boldsymbol{\theta}(i)}$ holds for any value of $\boldsymbol{a}_i(j)$. Thus, since j is arbitrary, we have $\boldsymbol{a}_i = (\boldsymbol{b}_i \boldsymbol{S}_{in})_{\geq \boldsymbol{\theta}(i)}$.

Conversely, suppose $\boldsymbol{a}_i = (\boldsymbol{b}_i \boldsymbol{S}_{in})_{\geq \boldsymbol{\theta}(i)}$ holds and $\boldsymbol{b}_i$ is a binary vector. So the third term in (1) is zero. Now suppose $\boldsymbol{a}_i(j) = 0$ for j ($1 \leq j \leq L$). Then $(\boldsymbol{b}_i \boldsymbol{S}_{in})(j)_{\geq \boldsymbol{\theta}(i)} = 0$ which implies $(\boldsymbol{b}_i \boldsymbol{S}_{in})(j) < \boldsymbol{\theta}(i)$. So in the state $\boldsymbol{S}_{in}(:,j)$, the number of true literals in C_i is less than the number of all literals in C_i. In other words, there is a false literal in C_i, and hence the number of false literals in C_i is greater than 1, thereby giving $\min_1(\boldsymbol{b}_i([\mathbb{1};\mathbb{1}] - \boldsymbol{S}^d_{in}))(j) = 1$. Consequently we conclude that when $\boldsymbol{a}_i(j) = 0$, $(\mathbf{1} - \boldsymbol{a}_i)(j) \cdot (\mathbf{1} - \min_1(\boldsymbol{b}_i([\mathbb{1};\mathbb{1}] - \boldsymbol{S}^d_{in})))(j) = 0$ holds. Therefore, since j is arbitrary, the first term in (1) is zero. Similarly by considering the case of $\boldsymbol{a}_i(j) = 1$, we can prove the second term in (1) is also zero. The third term is zero because $\boldsymbol{b}_i$ is binary. Hence we have $J^{\mathrm{and}}_i(\boldsymbol{b}_i) = 0$.

□

Dually to Proposition 1, we can prove

Proposition 2. *Suppose node i is an OR node. Then $J^{\mathrm{or}}_i(\boldsymbol{b}_i) = 0$ if-and-only-if $\boldsymbol{b}_i$ is a binary vector satisfying $\boldsymbol{a}_i = (\boldsymbol{b}_i \boldsymbol{S}_{in})_{\geq \boldsymbol{\theta}(i)}$ where $\boldsymbol{\theta}(i) = 1$.*

It follows from Proposition 1 and Proposition 2 that an AND/OR BN having the specified state transitions from $\boldsymbol{s}^{in}_j$ to $\boldsymbol{s}^{out}_j$ for j ($1 \leq j \leq L$) is obtained by learning $\boldsymbol{Q} = [\boldsymbol{b}_1; \cdots; \boldsymbol{b}_N]$ such that $J_i(\boldsymbol{b}_i) = 0$ ($J_i \in \{J^{\mathrm{and}}_i, J^{\mathrm{or}}_i\}$) for every i ($1 \leq i \leq N$).

4.2 Jacobians and a Learning Algorithm

We minimize $J_i \in \{J^{\mathrm{and}}_i, J^{\mathrm{or}}_i\}$ by Newton's method. So we compute two Jacobians, $\partial J^{\mathrm{and}}_i / \partial \boldsymbol{b}_i$ and $\partial J^{\mathrm{or}}_i / \partial \boldsymbol{b}_i$ as follows.

$$
\begin{aligned}
&\boldsymbol{J}^{\mathrm{and}}_{i\_Jacob} \qquad (3)\\
&= \partial J^{\mathrm{and}}_i / \partial \boldsymbol{b}_i \\
&= -((\mathbf{1} - \boldsymbol{a}_i) \odot (\boldsymbol{c}_i)_{<1})([\mathbb{1};\mathbb{1}] - \boldsymbol{S}^d_{in})^T \\
&\quad + (\boldsymbol{a}_i([\mathbb{1};\mathbb{1}] - \boldsymbol{S}^d_{in})^T) \odot \boldsymbol{b}_i + \boldsymbol{b}_i \odot (\mathbf{1} - \boldsymbol{b}_i) \odot (\mathbf{1} - 2\boldsymbol{b}_i)
\end{aligned}
$$

$$
\begin{aligned}
&\boldsymbol{J}^{\mathrm{or}}_{i\_Jacob} \qquad (4)\\
&= \partial J^{\mathrm{or}}_i / \partial \boldsymbol{b}_i \\
&= -(\boldsymbol{a}_i \odot (\boldsymbol{d}_i)_{<1}) \boldsymbol{S}^T_{in} \\
&\quad + ((\mathbf{1} - \boldsymbol{a}_i) \boldsymbol{S}^T_{in}) \odot \boldsymbol{b}_i + \boldsymbol{b}_i \odot (\mathbf{1} - \boldsymbol{b}_i) \odot (\mathbf{1} - 2\boldsymbol{b}_i)
\end{aligned}
$$

Here $\boldsymbol{c}_i = \boldsymbol{b}_i([\mathbb{1};\mathbb{1}] - \boldsymbol{S}^d_{in})$ and $\boldsymbol{d}_i = \boldsymbol{b}_i \boldsymbol{S}^d_{in}$. $\boldsymbol{c}_i(j)$ (resp. $\boldsymbol{d}_i(j)$) is the number of literals in a conjunction (resp. disjunction) associated with node i which are false (resp. true) in the state $\boldsymbol{S}_{in}(:,j) (1 \leq j \leq L)$.

We show the derivation of $\boldsymbol{J}^{\mathrm{and}}_{i\_Jacob}$. First to simplify notation, introduce $\boldsymbol{S}^{\neg d}_{in} = [\mathbb{1};\mathbb{1}] - \boldsymbol{S}^d_{in}$ which is the bit inversion of $\boldsymbol{S}^d_{in}$. Let $\boldsymbol{b}_i(p)$ be the p-th component of $\boldsymbol{b}_i$ and put $\partial \boldsymbol{b}_i / \partial \boldsymbol{b}_i(p) = \boldsymbol{I}_p$. $\boldsymbol{I}_p$ is a one-hot vector whose component is 0 except for the p-th component which is 1. Now we have

$$
\begin{aligned}
&\partial J^{\mathrm{and}}_i / \partial \boldsymbol{b}_i(p) \\
&= ((\mathbf{1} - \boldsymbol{a}_i) \bullet (-((\boldsymbol{c}_i)_{<1} \odot (\boldsymbol{I}_p \boldsymbol{S}^{\neg d}_{in}))) \\
&\quad + (\boldsymbol{a}_i \bullet ((\boldsymbol{I}_p \odot \boldsymbol{b}_i) \boldsymbol{S}^{\neg d}_{in})) \\
&\quad + ((\boldsymbol{b}_i \odot (\mathbf{1} - \boldsymbol{b}_i)) \bullet (\boldsymbol{I}_p \odot (\mathbf{1} - 2\boldsymbol{b}_i))) \\
&= (-((\mathbf{1} - \boldsymbol{a}_i) \odot (\boldsymbol{c}_i)_{<1})(\boldsymbol{S}^{\neg d}_{in})^T \\
&\quad + (\boldsymbol{a}_i (\boldsymbol{S}^{\neg d}_{in})^T) \odot \boldsymbol{b}_i + \boldsymbol{b}_i \odot (\mathbf{1} - \boldsymbol{b}_i) \odot (\mathbf{1} - 2\boldsymbol{b}_i) \bullet \boldsymbol{I}_p)
\end{aligned}
$$

Since p is arbitrary, we obtain (3). Here we use the fact that $(\boldsymbol{u} \bullet (\boldsymbol{v} \odot \boldsymbol{w})) = ((\boldsymbol{u} \odot \boldsymbol{v}) \bullet \boldsymbol{w})$ and $(\boldsymbol{u} \bullet (\boldsymbol{v}\boldsymbol{A})) = ((\boldsymbol{u}\boldsymbol{A}^T) \bullet \boldsymbol{v})$ hold for vectors $\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}$ and matrix $\boldsymbol{A}$. $\boldsymbol{J}^{\mathrm{or}}_{i\_Jacob}$ is similarly derived.

The updating formula for $\boldsymbol{b}_i$ depends on the node type of node i but it is uniformly described as

$$
\boldsymbol{b}_i \quad \leftarrow \quad \boldsymbol{b}_i - (J_i / \|\boldsymbol{J}_{i\_Jacob}\|^2_F) \boldsymbol{J}_{i\_Jacob}. \qquad (5)
$$

Here $J_i = J^{\mathrm{and}}_i$ and $\boldsymbol{J}_{i\_Jacob} = \boldsymbol{J}^{\mathrm{and}}_{i\_Jacob}$, or $J_i = J^{\mathrm{or}}_i$ and $\boldsymbol{J}_{i\_Jacob} = \boldsymbol{J}^{\mathrm{or}}_{i\_Jacob}$ depending on the node type. The updating formula (5) implements Newton's method and is derived from the first order Taylor expansion of J_i and by solving $J_i + (\boldsymbol{J}_{i\_Jacob} \bullet (\boldsymbol{b}_{i\_\mathrm{new}} - \boldsymbol{b}_i)) = 0$ w.r.t. $\boldsymbol{b}_{i\_\mathrm{new}}$ where $(\boldsymbol{a} \bullet \boldsymbol{b})$ is the inner product of $\boldsymbol{a}$ and $\boldsymbol{b}$.

Having derived the updating formula (5), we next describe **Algorithm 1**, an algorithm for learning an AND/OR BN in vector spaces. This algorithm learns a matricized AND/OR BN by minimizing a cost function $\sum_{i=1}^N J_i$ (possibly to zero). **Algorithm 1** is mostly (and hopefully) self-evident. There we use 1-norm $\|\boldsymbol{a}\|_1$ to denote the number of 1's in a binary vector $\boldsymbol{a}$ for convenience. Minimization is carried out by the inner q-loop and the outer p-loop exists for retry when the inner loop fails to achieve an error $\epsilon_i = 0$. Line 20 adds perturbation to $\boldsymbol{b}_i$ to escape a local minimum.

Since line 4 and line 8 include somewhat complicated operations behind them, we detail them. At line 4, a node type is determined. Given $\boldsymbol{b}_i$, we uniformly split an interval $[\min(\boldsymbol{b}_i) \max(\boldsymbol{b}_i)]$ into 20 levels $\{\mu_1, \ldots, \mu_{20}\}$ and generate 20 binary vectors $\boldsymbol{b}^*_{i,j} = (\boldsymbol{b}_i)_{\geq \mu_j}$ ($1 \leq j \leq 20$). For each j, we compute an error $\epsilon_{i,j} = \|\boldsymbol{a}_i - (\boldsymbol{b}^*_{i,j} \boldsymbol{S}^d_{in})_{\geq \boldsymbol{\mu}(i,j)}\|_1$ for two threshold levels, $\boldsymbol{\mu}(i,j) = \|\boldsymbol{b}^*_{i,j}\|_1$ and $\boldsymbol{\mu}(i,j) = 1$. If $\boldsymbol{\mu}(i,j) = \|\boldsymbol{b}^*_{i,j}\|_1$ yields a smaller $\epsilon_{i,j}$, node i is an AND node. Else it is an OR node. Finally, after deciding the node type with $\epsilon_{i,j}$ for every j, we choose j that gives the minimum $\epsilon_{i,j}$ and decide the node type as the one for j. At line 8, a thresholding operation is performed. Like line 4, we first compute thresholded vectors $\boldsymbol{b}^*_{i,j} = (\boldsymbol{b}_i)_{\geq \mu_j}$ for every j ($1 \leq j \leq 20$). Then we choose the best $\boldsymbol{b}^*_{i,j}$ as $\boldsymbol{b}^*_i$ that gives a minimum error $\epsilon_{i,j} = \|\boldsymbol{a}_i - (\boldsymbol{b}^*_{i,j} \boldsymbol{S}^d_{in})_{\geq \boldsymbol{\theta}(i,j)}\|_1$ where $\boldsymbol{\theta}(i,j) = \|\boldsymbol{b}^*_{i,j}\|_1$ if the node type is AND. Otherwise $\boldsymbol{\theta}(i,j) = 1$.

With max_try and max_itr being fixed, since there is only vector-matrix multiplication in the q-loop, time complexity per iteration in the q-loop is estimated as $O(N^2 L)$.

Algorithm 1 for learning an AND/OR BN as matrix

Input: $N \times L$ binary matrices $\boldsymbol{S}_{in}$, $\boldsymbol{S}_{out}$
Output: $N \times 2N$ binary matrix $\boldsymbol{Q}$, N dim. threshold vector $\boldsymbol{\theta}$
s.t. $\boldsymbol{S}_{out} = (\boldsymbol{Q}\boldsymbol{S}^d_{in})_{\geq \boldsymbol{\theta}}$ where $\boldsymbol{S}^d_{in} = [\boldsymbol{S}_{in}; (\mathbb{1} - \boldsymbol{S}_{in})]$

for $i = 1$ **to** N **do**
  randomly initialize $2N$ dim. row vector $\boldsymbol{b}_i$
  for $p = 1$ **to** max_try **do**
    determine node_type (AND,OR) of node i
    for $q = 1$ **to** max_itr **do**
      compute J_i (1),(2) and $\boldsymbol{J}_{i\_Jacob}$ (3),(4)
      update $\boldsymbol{b}_i$ by (5)
      threshold $\boldsymbol{b}_i$ to a binary vector $\boldsymbol{b}^*_i$
      set $\boldsymbol{Q}(i,:) = \boldsymbol{b}^*_i$
      if node_type = AND **then**
        put $\boldsymbol{\theta}(i) = \|\boldsymbol{b}^*_i\|_1$
      else
        $\boldsymbol{\theta}(i) = 1$
      end if
      compute $\epsilon_i = \|\boldsymbol{a}_i - (\boldsymbol{b}^*_i \boldsymbol{S}^d_{in})_{\geq \boldsymbol{\theta}(i)}\|_1$
      if $\epsilon_i = 0$ **then**
        exit p-loop
      end if
    end for
    $\boldsymbol{b}_i = 0.5 \cdot \boldsymbol{b}_i + 0.5 \cdot \Delta$ $\%\Delta \sim$ U(0,1)
  end for
end for
return $\boldsymbol{Q}$ and $\boldsymbol{\theta}$

5 Experiments

5.1 Random AND/OR BN: Learning Time vs Network Size

Here we measure learning time of **Algorithm 1** for random AND/OR BNs while varying network size which is measured by the number of nodes N.[8] For a given N, a max_indegree $\gamma = 5$ and a probability $p = 0.5$, we first generate an $N \times 2N$ binary matrix $\boldsymbol{Q}_0$ and a threshold vector $\boldsymbol{\theta}_0$ encoding a random AND/OR BN such that the indegree of a node[9] is uniformly distributed over $[1, \ldots, \gamma]$ and almost half of the nodes are AND nodes. Using this $\boldsymbol{Q}_0$, we generate randomly $L = 100$ state vectors whose element is one with probability p, store them in an $N \times L$ matrix $\boldsymbol{S}_{in} = [\boldsymbol{s}^{in}_1 \cdots \boldsymbol{s}^{in}_L]$ and compute the next state vectors $\boldsymbol{S}_{out} = (\boldsymbol{Q}_0 \boldsymbol{S}^d_{in})_{\geq \boldsymbol{\theta}_0} = [\boldsymbol{s}^{out}_1 \cdots \boldsymbol{s}^{out}_L]$ corresponding to $\boldsymbol{S}_{in}$. Finally we add noise to $\boldsymbol{S}_{out}$ by flipping each bit of $\boldsymbol{S}_{out}$ with a probability $q = 0.0/0.1$ to obtain $\boldsymbol{S}'_{out}$.

Using $\boldsymbol{S}_{in}$ and $\boldsymbol{S}'_{out}$ as learning data, we learn $\boldsymbol{Q}$ and $\boldsymbol{\theta}$ by **Algorithm 1** and measure learning time and error for various N. Learning error is computed as error = $|\boldsymbol{S}_{out} - (\boldsymbol{Q}\boldsymbol{S}^d_{in})_{\geq \boldsymbol{\theta}}|/N \cdot L$ which is the ratio of the total number of incorrectly predicted bits by the learned matrix $\boldsymbol{Q}$ to the total number of bits in $\boldsymbol{S}_{out}$.

N	1000	2000	3000	4000	5000
			$q = 0.0$		
time(s)	17.45	90.6	217.2	516.2	940.8
error(%)	0	0	0	0	0
			$q = 0.1$		
time(s)	539.2	1260.3	2170.6	3414.8	4532.5
error(%)	4.61	2.80	2.07	1.72	1.34

Table 1: Learning time and error for various network size N

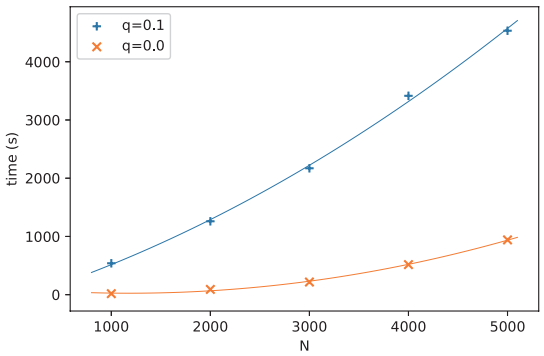


Figure 3: Learning time vs network size N with quadratic fit curves

The results are shown in Table 1 and Figure 3. Notice that learning time is quadratic w.r.t. N as indicated by quadratic regression in Figure 3, and this is what is expected from $O(N^2 L)$ time complexity of **Algorithm 1**. Also notice error = 0 is achieved for all N's when the noise probability q is zero. It means that the learned BN completely recovers randomly chosen 100 state transitions in the original AND/OR BN for all N's. Even if noise is added with $q = 0.1$ to the learning data $\boldsymbol{S}_{out}$, and hence 10% of the learning data is incorrect, the error by the learned BN is mostly less than 3%.

We next compare our method with the standard BN reconstruction methods: the BestFit extension algorithm (Lähdesmäki, Shmulevich, and Yli-Harja 2003) and the REVEAL algorithm (LIANG, FUHRMAN, and SOMOGYI 1998). However since they did not work in the setting where $N \geq 1000$, we used smaller data set ($N = 10 \ldots 90$). Figure 4 shows that our method is far more scalable with respect to the number of variables than those conventional ones. So we can say that our method is particularly effective in situations involving a large number of variables, e.g. genome-wide network analysis. We remark that all algorithms run in a single thread in this experiment for comparison.

5.2 Prediction Accuracy with Artificial Data

The previous experimental result suggests that the learned model, $\boldsymbol{Q}$, is likely to recover 100% of learning data as long

[8] This experiment and the next experiment are carried out using GNU Octave 4.2.2 on a PC with Intel(R) Xeon(R) CPU E5-2699 v4 @ 2.20GHz with 64 threads. All other experiments are carried out using GNU Octave 4.2.2 and Python 3.6.3 on a PC with Intel(R) Core(TM) i7-3770@3.40GHz CPU, 28GB memory.

[9] The indegree of a node n is the number of nodes having an outgoing edge to n. So, if the indegree of the node is k, the next state of n is determined by a Boolean formula comprised of k variables.

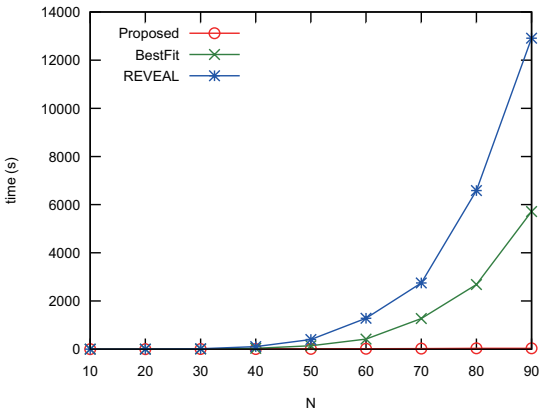


Figure 4: Comparison of learning time with conventional methods

as it is generated by an AND/OR BN, which naturally gives rise to the concern of overfitting. So in this experiment, using artificially generated state transition data, we examine prediction performance of the learned AND/OR BN in terms of 10-fold cross-validation (CV) used in the machine learning field.

First we generate state transition data $\{\boldsymbol{S}_{in}^{CV}, \boldsymbol{S}_{out}^{CV}\}$ from an AND/OR BN for $L \in \{50, 100, 150, 200\}$ where L is the length of state transitions as follows. Put $N = 100$. We construct a binary random matrix $\boldsymbol{Q}_0$ and a threshold vector $\boldsymbol{\theta}_0$ for an AND/OR BN BN_0 having N nodes in the same way as in the previous subsection. Then for each f ($\in [1 \dots 10]$), we simulate L state transitions $\boldsymbol{s}_0 \to \boldsymbol{s}_1 \to \cdots \to \boldsymbol{s}_L$ in BN_0 by starting from an initial state vector $\boldsymbol{s}_0$ whose element is one with probability $p = 0.5$ and sequentially computing $\boldsymbol{s}_j = (\boldsymbol{Q}_0 \boldsymbol{s}_{j-1}^d)_{\geq \boldsymbol{\theta}_0}$ $(1 \leq j \leq L)$. Then we put state vectors together as $\boldsymbol{S}_{in}^{(f)} = [\boldsymbol{s}_0 \cdots \boldsymbol{s}_{L-1}]$ and $\boldsymbol{S}_{out}^{(f)} = [\boldsymbol{s}_1 \cdots \boldsymbol{s}_L]$. $\boldsymbol{S}_{in}^{(f)}$ and $\boldsymbol{S}_{out}^{(f)}$ are $N \times L$ matrices.

Finally we concatenate all those matrices into one big $N \times 10 \cdot L$ matrix like $\boldsymbol{S}_{in}^{CV} = [\boldsymbol{S}_{in}^{(1)} \cdots \boldsymbol{S}_{in}^{(10)}]$. Similarly we construct $\boldsymbol{S}_{out}^{CV} = [\boldsymbol{S}_{out}^{(1)} \cdots \boldsymbol{S}_{out}^{(10)}]$. So $\boldsymbol{S}_{in}^{CV}$ and $\boldsymbol{S}_{out}^{CV}$ include transition data starting form 10 different initial states. Then, we measure prediction accuracy of the AND/OR BN learned from $\{\boldsymbol{S}_{in}^{CV}, \boldsymbol{S}_{out}^{CV}\}$ by **Algorithm 1** in terms of 10-fold CV.

Table 2 presents measured prediction accuracy "acc" with standard deviation for each L together with "acc_3" and "ratio_3". "acc_3" is the prediction accuracy restricted to nodes with a *small indegree*. Here *small indegree* means less than or equal to three. "ratio_3" is the ratio of the number of such nodes to N, the total number of nodes.

What is noticeable in Table 2 is high prediction accuracy for all L's. This is also true with small indegree (see "acc_3") and such nodes account for 26.4%(L=100) to 39.0%(L=200) of total nodes. The observation that small AND/ORs show good prediction performance will be repeated in the experiment with real data sets in the following Subsection 5.4.

The experimental result here demonstrates that as long as the data is generated by an AND/OR BN, prediction accuracy by **Algorithm 1** is very high, which however is not necessarily true of real data.

L	50	100	150	200
acc(%)	99.5(0.8)	99.9(0.1)	99.9(0.0)	99.8(0.0)
acc_3(%)	99.9(0.1)	99.9(0.1)	100(0.0)	100(0.0)
ratio_3(%)	36.4(2.3)	26.4(2.0)	29.4(2.3)	39.0(0.0)

Table 2: Prediction accuracy with state transition data

5.3 Learning Genome-wide AND/OR BNs for Budding Yeast

Here we address the problem of learning a *genome-wide* AND/OR BN from real data. By *genome-wide*, we mean analyzing the whole set of genes simultaneously, not learning a network of genes sampled from the original data.

We prepared two transcription profile data sets, E_MTAB01 and E_MTAB02[10], for budding yeast (Saccharomyces cerevisiae) obtained by measuring mRNA levels for three cell cycles. Each contains data on $N = 10,928$ labeled mRNA fractions measured at 41 time points. In the usual gene analysis, this transcription is reduced to gene expression profile by aggregating mRNA levels in the same gene to filter target genes and to reduce computational costs. In our setting however, to demonstrate scalability of our method, we uses all transcription profile in the data. In the sequel, we sometimes use interchangeably gene for mRNA for convenience.

We apply the robust multi-array average (RMA) method (Irizarry et al. 2003), a standard preprocessing method for micro-array data, and binarization by using an average threshold. As a result, we can consider each data set as state transition data containing 40 state transitions. Following Subsection 5.2, we construct two sets of $10,928 \times 40$ binary matrices $\boldsymbol{S}_{in}$ and $\boldsymbol{S}_{out}$, corresponding to E_MTAB01 and E_MTAB02, as learning data representing state transitions, and learn $\boldsymbol{Q}$ and $\boldsymbol{\theta}$ representing an AND/OR BN from them by **Algorithm 1**. Using learning parameters max_try = 10 and max_itr = 100 in **Algorithm 1**, we measure approximation accuracy[11] of the learned model in terms of 4-fold CV. Table 3 presents measured accuracies for each data set.

Looking at Table 3, we know that our approach achieves rather good approximation accuracy despite the simplicity

data_set	acc(%)	time(s)
E_MTAB01	87.3	104015.4
E_MTAB02	80.0	180253.0

Table 3: Approximation accuracy with genome-wide budding yeast data

[10] downloadable at `https://www.ebi.ac.uk/arrayexpress/experiments/E-MTAB-1908/`

[11] "approximation accuracy" means accuracy for learning data.

of AND/OR BN model. For example, acc = 87.3% for E_MTAB01 data set means that the next state of an mRNA is correctly recovered 87 times out of 100 trials on average by the learned model. Learning time however requires tens of hours[12] and there remain problems to be solved as we discuss next.

5.4 Constrained Learning and Relearning

In the previous subsection, we observe that real large scale biological data E_MTAB01 and E_MTAB02 are reasonably approximated by AND/OR BNs. However there remain two problems. One is low prediction accuracy. For example it turns out to be merely 40.9%, less than 50%, for E_MTAB01 data set by 4-fold CV.

The other is too long AND/OR formulas (some have more than 100 literals). Figure 5 is the indegree distribution of BN_E_MTAB01, an AND/OR BN learned from E_MTAB01 (cut off at indegree = 100). The indegree of a node is the number of literals in a conjunction or disjunction associated with it. From Table 4, we can see about 3,000 nodes have long AND/OR formulas containing more than 10 literals. Needless to say, such long AND/OR formulas are not very comprehensible to humans and worse, likely to cause overfitting.

Hence, we introduce two strategies working in tandem to improve readability and prediction accuracy. One is to constrain the length of AND/OR formulas when learning. The other is relearning an AND/OR formula for each gene after learning. First we constrain the length of AND/OR formulas to four or less. That is, when we construct an AND/OR formula at line 8 in **Algorithm 1**, we simply keep the initial (at most) four literals among those represented by $\boldsymbol{b}_i^*$ and pad zeros into the rest of $\boldsymbol{b}_i^*$.

Let $\mathtt{dom}(g)$ be a list of variables in a conjunction/disjunction associated with a gene g and $|\mathtt{dom}(g)|$ the number of variables in $\mathtt{dom}(g)$. Initially variables in $\mathtt{dom}(g)$ are selected from 10,928 ones by learning and $|\mathtt{dom}(g)|$ may

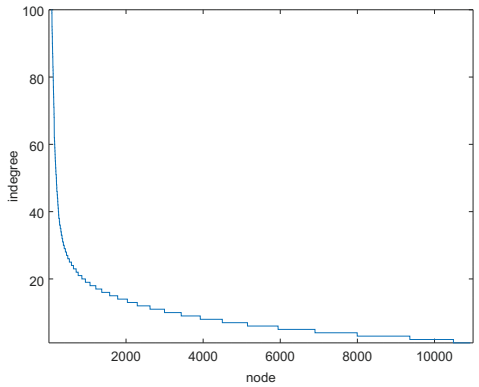


Figure 5: Indegree distribution of BN_E_MTAB01

[12]This is partly due to the use of a high-level language, Octave, and can be improved by shifting to a procedural language such as C++.

indegree x	$x \leq 3$	$4 \leq x \leq 10$	$x > 10$
#node	2933	4998	2997

Table 4: Indegree of node in BN_E_MTAB01

be 10 or 100 as seen in Figure 5 but now by this constraint, we have forcibly $|\mathtt{dom}(g)| \leq 4$.

Next to mitigate the negative effect of the forced condition $|\mathtt{dom}(g)| \leq 4$, we introduce a relearning process, i.e., learning again an AND/OR formula for each gene by **Algorithm 1**[13]. This time, however, learning data becomes part of $\boldsymbol{S}_{in}$ and $\boldsymbol{S}_{out}$ related to the gene g, that is $\boldsymbol{S}_{in}(\mathtt{dom}(g), :)$ and $\boldsymbol{S}_{out}(g, :)$.[14]

We conduct a learning experiment with E_MTAB01 and E_MTAB02 using the modified AND/OR BN learning described above and obtain Table 5. As we see, in the case of E_MTAB01, by adding size constraint and relearning, prediction accuracy by 4-fold CV is improved from 40.9% to 84.3%, more than doubled, and the learned AND/OR formula consists of 3.7 literals on average. Furthermore, if we focus on "acceptable AND/ORs" which are learned AND/OR formulas giving 0 or 1 error in the test phase of a fold in cross-validation, their average prediction accuracy in a fold further rises up to 94.9%. Note acceptable AND/OR formulas are not a minority of the learned AND/OR formulas. They are the majority occupying 6623.8/10928 = 60.6% of the learned AND/OR formulas on average. In other words, we could find, among the total 10,928 genes in the data set E_MTAB01, more than 6,000 genes together with associated small AND/OR formulas (consisting of less than or equal to four literals) which predict their next state with 94.9% accuracy. A similar result is obtained for E_MTAB02 where we could identify on average 3,797 genes and their associated small AND/OR formulas achieving 93.5% prediction accuracy.

What we have seen in this section is the genome-wide analysis of real data by an AND/OR BN, which, to our knowledge, is unprecedented. There are 10,928 genes and their transition to the next state can possibly depend on 10,928 genes. Despite the formidable number of possible genes to be considered in state transition, all possibilities are considered in a continuous space and their logi-

	AND/ORs		acceptable AND/ORs	
data_set	acc(%)	length	acc(%)	ratio
E_MTAB01	84.3	3.7	94.9	6623.8/10928
E_MTAB02	75.9	3.5	93.5	3797.0/10928

Table 5: Prediction accuracy of all AND/ORs and acceptable AND/ORs

[13]The hope is that the new set of variables might be further reduced or the choice of AND/OR might be properly adjusted.

[14]$\boldsymbol{S}_{in}(\mathtt{dom}(g), :)$ is a sub-matrix of $\boldsymbol{S}_{in}$ consisting of rows corresponding to $\mathtt{dom}(g)$, and $\boldsymbol{S}_{out}(g, :)$ is a row in $\boldsymbol{S}_{out}$ for the gene g.

cal interdependency is captured in the form of AND/OR formula. Also it is unexpected that small AND/OR formulas obtained through constrained learning and relearning achieve more than 90% prediction accuracy despite the fact that AND/OR Boolean formulas lack the expressive power of general Boolean ones.

6 Related Work

Boolean networks (BNs) express the logical dependency among nodes as a graph and AND/OR BNs aim at scalability by restricting general Boolean formulas to conjunction/disjunction. Concerning expressing dependency, dependency networks similarly express dependency by a graph but deal with probabilistic or statistical dependencies. For example Heckerman et al. proposed dependency networks for graphical model which are directed networks like Bayesian networks but allow cyclic paths (Heckerman et al. 2000). A consistent probability distribution is defined through the process of Gibbs sampling when it exists using a set of local conditional distributions. This sampling based approach is further extended to relational data (Schulte et al. 2016). Also there are statistical, correlation based dependency networks. Kenett et al. used partial correlation to define the influence of a node j on a node i through other nodes and applied to financial market analysis and immune system analysis (Kenett et al. 2010; Madi et al. 2011).

These dependency network approaches somehow recognize nodes that influence a target node. For example they are given as local conditional distributions in the case of (Heckerman et al. 2000) or they are deterministically computed (Kenett et al. 2010). Contrastingly in our approach, we identify the dependency relation among nodes by learning, i.e. by minimizing objective functions $J_i^{\mathrm{and}}(\cdot)$ and $J_i^{\mathrm{or}}(\cdot)$ (1), which empirically scales well with the number of nodes.

Technically, the scalability of our approach fundamentally depends on the matrix encoding of AND/OR BN in terms of an $N \times 2N$ matrix. It enables us first to formulate BN learning as a cost minimization problem in a continuous space and, second, to realize $O(N^2)$ time learning. This polynomial time/space complexity, though obtained at the cost of restricting general BNs to AND/OR BNs, is a key property to the success of our large scale AND/OR BN learning. It is also what differentiates our approach from previous matrix approaches where a $2N \times 2^N$ or $2^N \times 2^N$ matrix is used to express an unrestricted Boolean function (Cheng and Qi 2005; Cheng and Qi 2010; Kobayashi and Hiraishi 2014).

Restricted Boolean networks are another subclass of BNs other than AND/OR BNs in which the next state of a gene is determined by the difference between the weighted sum of "on" neighboring genes minus and that of "off" neighboring genes (Li et al. 2004; Higa, Louzada, and Hashimoto 2010; Ouyang et al. 2014). Heuristic algorithms inferring such restricted BNs have been proposed but they all work in a discrete space whose size is exponential in the number of nodes and scalability seems hard to achieve.

There are logical approaches to BN learning (Inoue, Ribeiro, and Sakama 2014; Tourret, Gentet, and Inoue 2017; Chevalier et al. 2019). From a logical point of view, our work is considered as a matricized version of "learning from interpretation transition" in logic programming in which a BN is represented as a propositional normal logic program (Inoue, Ribeiro, and Sakama 2014). The main difference is that we search for a solution by minimizing a differentiable cost function for scalability instead of applying logical operations such as resolution and subsumption in a symbolic space. Tourret et al. extracted DNF formulas from parameters of a feed-forward neural network learned from state transitions and convert them to logical rules describing a BN (Tourret, Gentet, and Inoue 2017).

Our learning is similar to deep neural network learning for biological data (Yue and Wang 2018) in that it is based on the sum-product of matrix and vector and the use of a nonlinear function. The difference is that we perform symbolic learning in vector spaces for Boolean formulas, not millions of network parameters for a classifier.

In bioinformatics, BNs have been used as a basic modeling tool for biological networks. For recent reviews, see (Hickman and Hodgman 2009; Liu 2015). From the viewpoint of gene-wide modeling of gene networks, however, most networks are relatively small. For example, Silvescu introduced Temporal Boolean networks (TBNs) where a state changes depending on the several past states and studied TBNs with 16 nodes using artificial data (Silvescu and Honavar 2001). Joo et al. proposed a BN with 5 nodes as a model for epithelial-to-mesenchymal transition (EMT) and examined dynamic stability of the cell attractors w.r.t. genetic mutations (Joo et al. 2018). Wilson et al. analyzed BNs with 5 nodes for gene regulatory network (GRN) to investigate the relationship between evolution and attractors (Wilson et al. 2019).

Kemmeren et al. performed genome-wide analysis of genetic perturbations by single gene deletions on Saccharomyces cerevisiae. They created a genetic perturbation network containing 3,476 nodes whose connectivity shows power-low distribution (Kemmeren et al. 2014).

Barman and Kwon inferred BNs by GA based on mutual information. They learned BNs with 100 nodes from artificial time-series data and also ones with 10 and 11 nodes from E.coli and yeast data (Barman and Kwon 2018).

Much larger networks are analyzed by Yang et al., though not by BNs. They synthesized GRNs by genetic expression programming (GEP), a variant of GA and GP, based on MapReduce framework and examined the acceleration effects on their parallel learning algorithm by parallel computing, using 500 genes selected from Saccharomyces cerevisiae (Yang et al. 2018).

7 Conclusion

We presented a novel approach to learning large scale AND/OR BNs. What differs most from the existing learning methods is that our learning is carried out in a vector space where an AND/OR BN is represented by a binary matrix $\boldsymbol{Q}$ paired with an integer vector $\boldsymbol{\theta}$ and they are learned from state transition data by minimizing a cost function expressed by matrix operations on $\boldsymbol{Q}$ and a thresholding operation using $\boldsymbol{\theta}$.

Using state transition data artificially generated from AND/OR BNs, we empirically confirmed $O(N^2)$ learning time behavior of our learning algorithm up to N=5,000 where N is the number of nodes in a BN. We also observed more than 99.5% prediction accuracy with artificial state transition data generated from an AND/OR BN with 100 nodes. Concerning applicability to real data, we learned two genome-wide AND/OR BNs consisting of $10,928$ nodes for budding yeast from transcription profiling data sets, each containing $10,928$ mRNAs and 40 transitions. The scale of BN learning in this size seems unprecedented. By introducing constrained learning together with relearning, we achieved for instance 84.3% prediction accuracy of the learned AND/OR BN for one data set and successfully extract more than 6,000 small AND/ORs whose average prediction accuracy reaches much higher 94.9%.

Our approach is expected to open a new way to scalable analysis of gene network data by scalable learning of BNs.

Acknowledgments

We thank Professor Katsumi Inoue for exciting discussions and useful information from a viewpoint of an expert on Boolean networks. This work was supported by JSPS KAKENHI Grant No.17H00763 and No.21H04905. This paper is also based on a part of results obtained from a project commissioned by the New Energy and Industrial Technology Development Organization (NEDO).

References

Akutsu, T.; Miyano, S.; and Kuhara, S. 1999. Identification of genetic networks from a small number of gene expression patterns under the boolean network model. In *Proceedings of the 4th Pacific Symposium on Biocomputing, PSB 1999*, 17–28.

Barman, S., and Kwon, Y.-K. 2018. A Boolean network inference from time-series gene expression data using a genetic algorithm. *Bioinformatics* 34(17):i927–i933.

Cheng, D., and Qi, H. 2005. Matrix expression of logic and fuzzy control. *Proceedings of the 44th IEEE Conference on Decision and Control* 3273–3278.

Cheng, D., and Qi, H. 2010. A linear representation of dynamics of boolean networks. *IEEE Transactions on Automatic Control* 55(10):2251–2258.

Chevalier, S.; Froidevaux, C.; Paulevé, L.; and Zinovyev, A. 2019. Synthesis of Boolean Networks from Biological Dynamical Constraints using Answer-Set Programming. In *31st International Conference on Tools with Artificial Intelligence*, ICTAI. IEEE.

Fogelberg, C., and Palade, V. 2009. *Machine Learning and Genetic Regulatory Networks: A Review and a Roadmap*, volume 201. 3–34.

Heckerman, D.; Chickering, D. M.; Meek, C.; Rounthwaite, R.; and Kadie, C. M. 2000. Dependency networks for inference, collaborative filtering, and data visualization. *J. Mach. Learn. Res.* 1:49–75.

Hickman, G., and Hodgman, C. 2009. Inference of gene regulatory networks using boolean-network inference methods. *Journal of bioinformatics and computational biology* 7:1013–29.

Higa, C.; Louzada, V.; and Hashimoto, R. 2010. Analysis of gene interactions using restricted boolean networks and time-series data. volume 6053, 61–76. Springer-Verlag New York, Inc.

Inoue, K.; Ribeiro, T.; and Sakama, C. 2014. Learning from interpretation transition. *Machine Learning* 94(1):51–79.

Irizarry, R. A.; Hobbs, B.; Collin, F.; Beazer-Barclay, Y. D.; Antonellis, K. J.; Scherf, U.; and Speed, T. P. 2003. Exploration, normalization, and summaries of high density oligonucleotide array probe level data. *Biostatistics* 4(2):249–264.

Joo, J. I.; Zhou, J. X.; Huang, S.; and Cho, K.-H. 2018. Determining relative dynamic stability of cell states using boolean network model. *Scientific Reports* 8.

Kauffman, S. A. 1993. *The Origins of Order Self-Organization and Selection in Evolution*. Oxford University Press.

Kemmeren, P.; Sameith, K.; van de Pasch, L. A.; Benschop, J. J.; Lenstra, T. L.; Margaritis, T.; O'Duibhir, E.; Apweiler, E.; van Wageningen, S.; Ko, C. W.; van Heesch, S.; Kashani, M. M.; Ampatziadis-Michailidis, G.; Brok, M. O.; Brabers, N. A.; Miles, A. J.; Bouwmeester, D.; van Hooff, S. R.; van Bakel, H.; Sluiters, E.; Bakker, L. V.; Snel, B.; Lijnzaad, P.; van Leenen, D.; Koerkamp, M. J. G.; ; and Holstege, F. C. 2014. Large-scale genetic perturbations reveal regulatory networks and an abundance of gene-specific repressors. *Cell* 157(3):740–752.

Kenett, D. Y.; Tumminello, M.; Madi, A.; Gur-Gershgoren, G.; Mantegna, R. N.; and Ben-Jacob, E. 2010. Dominating clasp of the financial sector revealed by partial correlation analysis of the stock market. *PloS one* 5(12).

Kobayashi, K., and Hiraishi, K. 2014. Ilp/smt-based method for design of boolean networks based on singleton attractors. *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 11:1253–1259.

Li, F.; Long, T.; Lu, Y.; Ouyang, Q.; and Tang, C. 2004. The yeast cell-cycle network is robustly designed. *Proceedings of the National Academy of Sciences of the United States of America* 101:4781–6.

LIANG, S.; FUHRMAN, S.; and SOMOGYI, R. 1998. Reveal, a general reverse engineering algorithm for inference of genetic network architectures. In *Pacific Symposium on Biocomputing*, volume 3, 18–29.

Liu, Z.-P. 2015. Reverse engineering of genome-wide gene regulatory networks from gene expression data. *Current genomics* 16(1):3–22.

Lähdesmäki, H.; Shmulevich, I.; and Yli-Harja, O. 2003. On learning gene regulatory networks under the boolean network model. In *Machine Learning*, 147–167.

Madi, A.; Kenett, D.; Bransburg Zabary, S.; Boccaletti, S.; Tauber, A.; Cohen, I.; and Ben-Jacob, E. 2011. Analyses of

antigen dependency networks unveil immune system reorganization between birth and adulthood. *Chaos* 21:016109.

Martin, S.; Zhang, Z.; Martino, A.; and Faulon, J.-L. 2007. Boolean dynamics of genetic regulatory networks inferred from microarray time series data. *Bioinformatics* 23(7):866–874.

Melkman, A.; Tamura, T.; and Akutsu, T. 2010. Determining a singleton attractor of an and/or boolean network in o(1.587^n) time. *Information Processing Letters* 110:565–569.

Ouyang, H.; Fang, J.; Shen, L.; Dougherty, E.; and Liu, W. 2014. Learning restricted boolean network model by time-series data. *EURASIP journal on bioinformatics & systems biology* 2014:10.

Schulte, O.; Qian, Z.; Kirkpatrick, A.; Yin, X.; and Sun, Y. 2016. Fast learning of relational dependency networks. *Machine Learning* 103:377–406.

Silvescu, A., and Honavar, V. 2001. Temporal boolean network models of genetic networks and their inference from gene expression time series. In Wu, C.; Wang, P.; and Wang, J., eds., *Proceedings of the Atlantic Symposium on Computational Biology and Genome Information Systems and Technology, CBGIST 2001*, 260–265.

Tourret, S.; Gentet, E.; and Inoue, K. 2017. Learning human-understandable description of dynamical systems from feed-forward neural networks. In Cong, F.; Leung, A. C.; and Wei, Q., eds., *Advances in Neural Networks - ISNN 2017 - 14th International Symposium, Proceedings, Part I*, volume 10261 of *Lecture Notes in Computer Science*, 483–492. Springer.

Wilson, S.; James, S.; Whiteley, D.; and Krubitzer, L. 2019. Limit cycle dynamics can guide the evolution of gene regulatory networks towards point attractors. *Scientific Reports* 9:16750.

Xu, H.; Ang, Y.; Sevilla, A.; Lemischka, I.; and Ma'ayan, A. 2014. Construction and validation of a regulatory network for pluripotency and self-renewal of mouse embryonic stem cells. *PLoS computational biology* 10(6188).

Yang, B.; Bao, W.; Huang, D.-S.; and Chen, Y. 2018. Inference of large-scale time-delayed gene regulatory network with parallel mapreduce cloud platform. *Scientific Reports* 8.

Yue, T., and Wang, H. 2018. Deep learning for genomics: A concise overview. *ArXiv* abs/1802.00810.