Machine Learning, 52, 147–167, 2003

On Learning Gene Regulatory Networks Under the Boolean Network Model

HARRI LÄHDESMÄKI harri.lahdesmaki@tut.fi
Institute of Signal Processing, Digital Media Institute, Tampere University of Technology, P.O. Box 553, FIN-33101 Tampere, Finland

ILYA SHMULEVICH is@ieee.org
Cancer Genomics Laboratory, University of Texas M.D. Anderson Cancer Center, 1515 Holcombe Blvd., Box 85, Houston, TX 77030, USA

OLLI YLI-HARJA yliharja@cs.tut.fi
Institute of Signal Processing, Digital Media Institute, Tampere University of Technology, P.O. Box 553, FIN-33101 Tampere, Finland

Editors: Paola Sebastiani, Isaac S. Kohane and Marco F. Ramoni

Abstract. Boolean networks are a popular model class for capturing the interactions of genes and global dynamical behavior of genetic regulatory networks. Recently, a significant amount of attention has been focused on the inference or identification of the model structure from gene expression data. We consider the Consistency as well as Best-Fit Extension problems in the context of inferring the networks from data. The latter approach is especially useful in situations when gene expression measurements are noisy and may lead to inconsistent observations. We propose simple efficient algorithms that can be used to answer the Consistency Problem and find one or all consistent Boolean networks relative to the given examples. The same method is extended to learning gene regulatory networks under the Best-Fit Extension paradigm. We also introduce a simple and fast way of finding all Boolean networks having limited error size in the Best-Fit Extension Problem setting. We apply the inference methods to a real gene expression data set and present the results for a selected set of genes.

Keywords: gene regulatory networks, network inference, Consistency Problem, Best-Fit Extension paradigm

1. Introduction

A central focus of genomic research concerns understanding the manner in which cells execute and control the enormous number of operations required for normal function and the ways in which cellular systems fail in disease. In biological systems, decisions are reached by methods that are exceedingly parallel and extraordinarily integrated. An important goal is to understand the nature of cellular function and the manner in which genes and their products collectively form a biological system. In contrast to the reductionistic approaches in biology, it is becoming increasingly apparent that it is necessary to study the behavior of genes in a holistic rather than in an individual manner. Such approaches inevitably require computational and formal methods to process massive amounts of data, to understand general principles governing the system under study, and to make useful predictions about

system behavior in the presence of known conditions. A significant role is played by the development and analysis of mathematical and computational methods in order to construct formal models of genetic interactions. This research direction provides insight and a conceptual framework for an integrative view of genetic function and regulation and paves the way toward understanding the complex relationship between the genome and the cell.

A number of different approaches to gene regulatory network modeling have been introduced, including linear models (D'Haeseleer et al., 1999) Bayesian networks (Murphy & Main, 1999; Friedman et al., 2000; Hartemink et al., 2001), neural networks (Weaver, Workman, & Stormo, 1999; Vohradsky, 2001), differential equations (Chen, He, & Church, 1999; Mestl, Plahte, & Omholt, 1995), and models including stochastic components on the molecular level (McAdams & Arkin, 1997) (see Smolen, Baxter, & Byrne, 2000; Hasty et al., 2001; de Jong, 2002) for reviews of general models). A model class that has received a considerable amount of attention is the *Boolean network* (BN) model originally introduced by Kauffman (Kauffman, 1969; Glass & Kauffman, 1973). Good reviews can be found in Huang (1999), Kauffman (1993), and Somogyi and Sniegoski (1996). In this model, the state of a gene is represented by a Boolean variable (ON or OFF) and interactions between the genes are represented by Boolean functions, which determine the state of a gene on the basis of the states of some other genes. Recent work suggests that even when gene expression data are analyzed entirely in the binary domain (only two quantization levels), meaningful biological information can be successfully extracted (Shmulevich & Zhang, 2002c; Tabus, Rissanen, & Astola, 2002). One of the appealing properties of BNs is that they are inherently simple, emphasizing generic network behavior rather than quantitative biochemical details, but are able to capture much of the complex dynamics of gene regulatory networks.

Most of the recent work on Boolean networks has focused on identifying the structure of the underlying gene regulatory network from gene expression data (Liang, Fuhrman, & Somogyi, 1998; Akutsu et al., 1998; Akutsu, Miyano, & Kuhara, 1999; Ideker, Thorsson, & Karp, 2000; Karp, Stoughton, & Yeung, 1999; Maki et al., 2001; Noda et al., 1998; Shmulevich et al., 2002b). A related issue is to find a network that is consistent with the given observations or determine whether such a network exists at all. This is known as the *Consistency Problem* (see Section 3.1). The Consistency Problem has been addressed and algorithms solving the problem have been introduced in Akutsu et al. (1998) and Akutsu, Miyano, and Kuhara (1999).

On the other hand, one may argue that the simple Consistency Problem can not be used to infer a network from real data. That is, due to the complex measurement process, ranging from hybridization conditions to image processing techniques, expression patterns exhibit uncertainty. For example, in the case of cDNA microarray data, it is widely recognized that reproducibility of measurements and between-slide variation is a major issue (Chen, Dougherty, & Bittner, 1997; Kerr et al., 2002). Furthermore, genetic regulation exhibits considerable uncertainty on the biological level. Indeed, evidence suggests that this type of "noise" is in fact advantageous in some regulatory mechanisms (McAdams & Arkin, 1999).

A survey of implicit underlying assumptions made in Boolean network inference and some potential noise sources are discussed in Yli-Harja, Linne, and Astola (2001). The implicit assumption of noise-free measurements, made in most of the methods presented thus far, may lead to unpredictable results. In order to cope with noisy expression patterns,

Akutsu, Miyano, and Kuhara (2000) have proposed a robust inference algorithm for so-called noisy Boolean networks. Shmulevich et al. showed that inferring a gene regulatory network using the so-called *Best-Fit Extension* method is polynomial-time solvable (Shmulevich et al., 2002b), therefore making that method computationally useful.

In this paper, we introduce a method that can be used to significantly accelerate the search for consistent gene regulatory networks under the Boolean network model. The same method, with minor changes, applies to learning gene regulatory networks under the Best-Fit Extension Problem. Two different classes of functions are considered in this paper: (i) the class of all Boolean functions and (ii) the class of Boolean functions containing no more than k variables.

This paper is organized as follows. Background information and necessary definitions are given in Section 2. The Consistency and Best-Fit Extension Problem are formulated in Section 3 and the proposed inference algorithms are then introduced in Section 4. Section 5 introduces a method for finding a set of Boolean functions having limited error-size. Experiments with real gene expression data are shown in Section 6 and, finally, discussion and concluding remarks are given in Section 7.

2. Background and definitions

For consistency of notation with other related work, we use similar notation as in Akutsu, Miyano, and Kuhara (1999) and Shmulevich et al. (2002b). In the learning part of this paper, however, the notation follows mainly the paper (Shmulevich et al., 2002b). Let us start by stating the definition of a Boolean network (BN).

2.1. Boolean networks

Definition 1. A Boolean network $G(V, B)$ is defined by a set of nodes (genes) $V = \{x_1, \ldots, x_n\}$ and a set of Boolean functions $B = \{f_1, \ldots, f_n\}$. We write simply $x_i = 1$ (resp. $x_i = 0$) to denote that the ith node (gene) is expressed (resp. not expressed). Each Boolean function $f_i(x_{i_1}, \ldots, x_{i_k})$ with k specific input nodes is assigned to node x_i and is used to update its value. The values of all the nodes in V are then updated synchronously.

Regulation of nodes is defined by the set B of Boolean functions. In detail, given the value of the nodes V at time t, the Boolean functions are used to update the value of the nodes at time $t+1$. The synchronous update process is then repeated making the network dynamic. In order to capture the dynamic nature of the network, it may be useful to consider the *wiring diagram* (see e.g. Akutsu, Miyano, & Kuhara, 1999; Shmulevich et al., 2002a, 2002b) which gives an explicit way of implementing the updating procedure. However, we omit these details here and just use the conceptual synchronous updating process in this paper.

In general, k, the number of input variables (genes) in each Boolean function, can vary as a function of i. However, without loss of generality, we can define k to be a constant equal to n and let the missing variables in each function be *fictitious*. A variable x_i in a Boolean function

f is fictitious if $f(x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n) = f(x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n)$ holds for all $x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n$. A variable is called *essential* if it is not fictitious.

The number of essential input variables in a Boolean function may vary from 0 to n. However, it has been shown (Akutsu, Miyano, & Kuhara, 1999) that exponential number of training samples, relative to n, are needed to infer a Boolean network without any constraints on k. In addition, real gene regulatory networks are shown to have moderately low mean connectivity. For instance, estimate of mean connectivity in *Escherichia coli* is around 2 or 3 (Thieffry et al., 1998) while in *higher metazoa*, it is between 4 and 8 (Arnone & Davidson, 1997). Further, preliminary results made by utilizing the Minimum Description Length (MDL) principle in gene regulatory network learning also agreed with the view that *indegree* (the number of variables in Boolean functions) of each gene could be "low" (Tabus & Astola, 2001). On the other hand, exact values of k for different genes are unknown. Therefore, we let the indegree k, as well as the number of genes n and the number of measurements m, be free parameters throughout this paper.

2.2. Asymptotic notation

Asymptotic notation is used to define the complexity of learning algorithms. Particularly, so-called *tight upper bound* notation is used in terms of n, m and k. The maximum indegree of functions, k, may be fairly small when compared to the number of nodes n. Thus, one could say that k acts as a constant in our analysis because asymptotic notation only applies in the limit, i.e., when all variables in the notation approach infinity. On the other hand, when considering the definition of the tight upper bound in Eq. (1) below, we can deduce that it holds for sufficiently large values of variables (see the role of n_0, m_0 and k_0 in Eq. (1)). In particular, k, which is intuitively and biologically the smallest one, has previously been tied into a super-exponential term 2^{2^k} (see e.g. Akutsu, Miyano, & Kuhara, 1999; Shmulevich et al., 2002b), making it relevant even for fairly small values in this context.

The definition of a Boolean network gives us an extra constraint: each node in a network can depend on at most n variables. This is also reflected in the asymptotic notation. So, let us briefly re-examine the definition of the tight upper bound with that extra constraint using the problem domain variables.

Definition 2. A function g is a tight upper bound for a function f, denoted as $f(n, m, k) \in O(g(n, m, k))$, if and only if there exist positive constants n_0, m_0, k_0 and c such that

$$(n > n_0, m > m_0, k > k_0, n \geq k) \Rightarrow 0 \leq f(n, m, k) \leq cg(n, m, k). \tag{1}$$

The above definition is usually given without the extra constraint $n \geq k$ such that it defines a *class of functions* $O(g(n, m, k))$, which contains all functions f satisfying the above conditions (Cormen, Leiserson, & Rivest, 1998).

3. Problem formulation

We now define the Consistency and Best-Fit Extension Problems first (Boros, Ibaraki, & Makino, 1998; Akutsu, Miyano, & Kuhara, 1999; Shmulevich et al., 2002b). Note that the

following definitions are given for one function (gene) only. Their extensions to Boolean networks can be defined by repeating the same definition for all functions in the network.

3.1. *Consistency Problem*

In the Consistency Problem (also called Extension Problem) we are concerned with establishing a Boolean function f from some class of functions $\mathcal{C}$ such that f correctly separates the given true and false examples, if one exists. In other words, the goal is to find a perfect Boolean classifier for the given binary examples.

A *partially defined Boolean function* pdBf(T, F) is defined by two sets, T and F, both belonging to $\{0, 1\}^n$, where T and F denote the set of true and false examples, respectively. For a Boolean function f, define the set of true and false vectors as $T(f) = \{x \in \{0, 1\}^n : f(x) = 1\}$ and $F(f) = \{x \in \{0, 1\}^n : f(x) = 0\}$. The function f is then said to be a *consistent extension* of pdBf(T, F) if $T \subseteq T(f)$ and $F \subseteq F(f)$. The Consistency Problem entails deciding whether or not there exists a consistent extension $f \in \mathcal{C}$ for the given sets of true and false examples T and F. In the affirmative case the function must also be returned. Thus, the Consistency Problem has an affirmative answer if and only if there exists at least one function $f \in \mathcal{C}$ that has consistent output for all examples (inputs). It is worth mentioning that we are also searching for all consistent functions in the following sections.

3.2. *Best-Fit Extension Problem*

As in the case of the Consistency Problem, let us assume that we are given two sets of binary vectors T and F, both belonging to $\{0, 1\}^n$, which define a *partially defined Boolean function* pdBf(T, F). Let us also assume that we are given positive weights $w(x) > 0$ for all $x \in T \cup F$ and define the weight of any subset $S \subseteq T \cup F$ to be

$$w(S) = \sum_{x \in S} w(x). \tag{2}$$

Now, the *error size* of function f is defined as

$$\varepsilon(f) = w(T \cap F(f)) + w(F \cap T(f)). \tag{3}$$

In the Best-Fit Extension Problem we are looking for Boolean functions from a certain class of functions $\mathcal{C}$ which make as few "misclassifications" as possible. The formal definition can be stated as follows. Find subsets T^* and F^* such that $T^* \cap F^* = \emptyset$ and $T^* \cup F^* = T \cup F$ for which the pdBf(T^*, F^*) has an extension in some class of functions $\mathcal{C}$ and so that $w(T^* \cap F) + w(F^* \cap T)$ is minimum. Then, any extension $f \in \mathcal{C}$ of pdBf(T^*, F^*) has minimum error size. (Also note that the Consistency Problem can be viewed as a special case of the Best-Fit Extension Problem when $\varepsilon(f) = 0$.) In Section 5 we extend this problem formulation to find all functions from $\mathcal{C}$ that have limited error size.

In practice, we may have measured identical true and/or false vectors several times, each of them associated possibly with different positive weight. Consider a simple case where

we have measured the same binary vector x three times, $x = x_1 = x_2 = x_3$, and all of them have the same weight $w(x) = w(x_1) = w(x_2) = w(x_3)$. Further, assume that $x_1 \in T$ and $x_2, x_3 \in F$. Then, what should be the output value for the input x? If we form the sets T and F without taking into account possible multiplicities, x will appear in both T and F with the same weight $w(x)$. In that case, it will not make any difference, according to Eqs. (2) and (3), if we choose the output value of $f(x)$ to be 0 or 1. On the other hand, we have measured x in F two times and in T only once (with the same weight). That should be an argument for the output value of the function with input x being 0. Since the weight of any subset $S \subseteq T \cup F$ is additive, as shown in Eq. (2), it is justified to define the weight of $x \in T$ (resp. F) to be the sum of all weights of vectors x belonging to T (resp. F). So, before forming the sets T and F, and assigning weights to their elements, one needs to take into account possible multiplicities in the original lists (or multisets) of measurements M. Thus, the weight of a particular vector $x \in T$ is defined as

$$w(x) = \sum_{x' \in M\,:\,(x'=x)\wedge(x' \in T)} w(x'). \tag{4}$$

A similar definition can be given for $x \in F$. Note, however, that the number of original measurements m may decrease by the above procedure since the multiple occurrences of the same vectors in T and/or F are removed. The new weights could be computed in advance. Alternatively, the effects of the new weights can be achieved, for example by embedding the weight assignment procedure into the Best-Fit inference method to be discussed in Section 4.5. Further, the complexity of the Best-Fit inference method would remain the same when the possible change in the value of m is ignored and therefore, we omit the weight assignment procedure from further complexity considerations. In order not to make the following discussion more complicated, we suppose that we are given the sets of true and false vectors, T and F, and the corresponding weights (recomputed according to Eq. (4) if needed). We also assume that the set of true and false examples consists of m examples, i.e., $|T| + |F| = m$.

4. Proposed inference methods

The inference methods are first introduced for functions having n variables, where n is the number of genes. The results are then reduced to the class of functions having no more than k variables ($0 \leq k \leq n$). Further, these results are then extended to Boolean networks, where each gene is associated with a Boolean function having no more than k variables, which is the case of primary interest. Note that k is still a free parameter ranging from 0 to n.

4.1. Consistency Problem, n variables

Let us first consider the class of all Boolean functions. It is relatively simple to see that in this class, a pdBf(T, F) has an extension if and only if T and F are disjoint. This can be checked in time $O(|T| \cdot |F| \cdot \text{poly}(n))$ by comparing all examples pair wise, where poly(n) is the time needed to examine one pair of examples (Boros, Ibaraki, & Makino, 1998).[1] It is

easy to see that, in the worst case, the asymptotic time consumption, when written in terms of m and n, is equal to $O(m^2 \cdot \text{poly}(n))$, since $|T| + |F| = m$.

Sorting algorithms provide an alternative method of solving the Consistency Problem. Let us first define a bijective mapping $s : \{0, 1\}^n \to \{1, \ldots, 2^n\}$ that can be used to encode all binary vectors of length n to positive integers. The corresponding inverse mapping is denoted as s^{-1}. Then, the Consistency Problem can be answered by encoding all binary vectors $x \in T \cup F$ by positive integers, sorting the integers, running through the sorted list, and testing whether there exists an example pattern that belongs to both T and F. The testing phase can be done simply by comparing consecutive elements in the sorted list. That is, if the sorted list is strictly increasing (or decreasing) then there is no consecutive terms in that list which have the same value. This further implies that T and F are disjoint, giving an affirmative answer to the Consistency Problem. So, asymptotic time consumption is now controlled by sorting, which can be solved in time $O(m \cdot \log m \cdot \text{poly}(n))$. This method can only be used for answering the Consistency Problem, but not for actually finding a consistent network.

Example 1. Assume that $n = 3$ and we have gathered five measurements as follows $T = \{001, 101, 110\}$ and $F = \{010, 101\}$. The encoding step produces the following sets of integers $s(T) = \{2, 6, 7\}$ and $s(F) = \{3, 6\}$. Sorting elements in $s(T)$ and $s(F)$ in increasing order gives the list (2, 3, 6, 6, 7), based on which a negative answer to the Consistency Problem can be given.

In general, instead of just deciding whether or not a consistent function exists, we are also interested in finding one or all consistent Boolean functions relative to the given examples. Each Boolean function f having n variables is completely defined by its *truth table* having 2^n entries. Define a 4-ary column vector $\mathbf{f} \in \{0, 1, ?, *\}^{2^n}$ that is used to encode the truth table of a Boolean function f having n variables. $\mathbf{f}$ is indexed from 1 to 2^n (indices are shown as subscripts $\mathbf{f}_i$). So, the ith element of $\mathbf{f}$ defines the binary output value for the input vector $s^{-1}(i)$. If $\mathbf{f}_i$ equals to ?, the output corresponding to the input $s^{-1}(i)$ is undetermined, or can be either 0 or 1. Further, $\mathbf{f}$ is assumed to be initialized to $\mathbf{f} = (?, \ldots, ?)$. (In practice, any constant-valued memory block is appropriate.) Let us call the non-binary truth table, possibly containing ?s, as the "generalized" truth table and for iteratively updating $\mathbf{f}$ define the jth version of $\mathbf{f}$ to be $\mathbf{f}^j$. For now, let us assume that all measurements are unique.[2] Then, the following procedure answers the Consistency Problem and finds all consistent functions relative to the given examples.

This procedure runs through all examples $x \in T \cup F$ and during every cycle updates $\mathbf{f}^j$ as

$$\mathbf{f}^j_{s(x)} = \begin{cases} 0, & \text{if } x \in F \wedge \mathbf{f}^{j-1}_{s(x)} = ? \\ 1, & \text{if } x \in T \wedge \mathbf{f}^{j-1}_{s(x)} = ?, \\ *, & \text{otherwise} \end{cases} \tag{5}$$

where $j = 1, \ldots, m$ is the index of the current cycle (measurement) and $\mathbf{f}^0$ refers to the initial setting of vector $\mathbf{f}$, i.e., $\mathbf{f}^0 = (?, \ldots, ?)$. It can be easily seen that if the last branch of

Eq. (5) is used, no consistent function can be found for the given examples. Otherwise there exists at least one consistent function and all of them are defined by **f** in the "generalized" form. Thus, when the time spent on memory initialization is ignored, the Consistency Problem can be answered and the "generalized" truth table is found in time

$$O((|T| + |F|) \cdot \text{poly}(n)) = O(m \cdot \text{poly}(n)). \tag{6}$$

Time complexity is, in the worst case, of the magnitude lower than $O(|T| \cdot |F| \cdot \text{poly}(n))$.

4.2. Forming standard truth tables

The found generalized truth table **f** may now contain question marks. As mentioned above, each ? specifies an undetermined element of the truth table **f**. If we prefer standard truth tables, each question mark must be changed to a single bit, either 0 or 1. In view of the Consistency Problem, each ? can be set to either 0 or 1 and the resulting truth tables, regardless of the chosen bits, are all consistent. Conversion to a standard truth table needs a run through the truth table **f** and simultaneously each ? has to be set to either 0 or 1. So, **f** can be converted to a standard truth table in time $O(2^n)$.

Let $|\mathbf{f}|_?$ denote the number of question marks in vector **f**. Then, the total number of consistent functions is $2^{|\mathbf{f}|_?}$. Correspondingly, conversion of all consistent functions into standard truth tables needs time $O(2^{|\mathbf{f}|_?} \cdot 2^n)$ during which one needs to keep track of the previously set bits such that all possible combinations are found. The number of consistent functions may become super-exponential relative to n, or exponential relative to the length of truth table 2^n. For instance, in the case of only one measurement ($m = 1$), there exist 2^{2^n-1} consistent functions. In such a case, we may be interested in converting only some number h of consistent functions, which takes time $O(h \cdot 2^n)$. Various additional criteria could then be imposed on the selection of h consistent functions. For example, we may be interested in producing h consistent functions, among all consistent functions, having minimal complexity.

4.3. Consistency Problem, k variables

Let us now focus on the class of Boolean functions having no more than k variables, where k is assumed to satisfy $0 \leq k \leq n$. The answer to the Consistency Problem and all consistent Boolean functions in the generalized form can be found simply by applying the above algorithm to all $\binom{n}{k}$ different variable combinations. The time needed for this is $O(\binom{n}{k} \cdot m \cdot \text{poly}(k))$.

Let us temporarily index all the found generalized truth tables (4-ary vectors), corresponding to $\binom{n}{k}$ different variable combinations, as $\mathbf{f}^j$, $j = 1, \ldots, \binom{n}{k}$. Question marks, in turn, can be removed in time $O(\binom{n}{k} \cdot 2^{|\mathbf{f}|_{?,\max}} \cdot 2^k)$, where $|\mathbf{f}|_{?,\max}$ is the maximum of all the $|\mathbf{f}^j|_?$ over all $\binom{n}{k}$ variable combinations, i.e., $|\mathbf{f}|_{?,\max} = \max_j |\mathbf{f}^j|_?$. Again, if we are interested in converting only some number h of consistent functions, that can be done in time $O(h \cdot 2^k)$.

4.4. *Consistency Problem, n genes and k variables*

To generalize this for Boolean networks, we must repeat the above process for all n genes, essentially multiplying the time needed for finding functions for one gene by n. Thus, the resulting algorithm for Boolean networks operates in time

$$O(\tbinom{n}{k} \cdot n \cdot m \cdot \text{poly}(k)), \tag{7}$$

where $0 \leq k \leq n$ (follows from Definition 1).

Let us again temporarily index all the found generalized truth tables (4-ary vectors) as $\mathbf{f}^{i,j}$, $i = 1, \ldots, n$, $j = 1, \ldots, \binom{n}{k}$. The generalized truth tables can be converted into standard type truth tables in time $O(\binom{n}{k} \cdot n \cdot 2^{|\mathbf{f}|_{?,\max}} \cdot 2^k)$ or $O(n \cdot h \cdot 2^k)$, where $|\mathbf{f}|_{?,\max}$ is the maximum of all the $|\mathbf{f}^{i,j}|_?$ over all n genes and $\binom{n}{k}$ variable combinations, i.e., $|\mathbf{f}|_{?,\max} = \max_{i,j} |\mathbf{f}^{i,j}|_?$. In principle, either of these terms should be added to Eq. (7) but it is ignored in this context since we aim to compare inference complexities with the previously published ones. Another reason for leaving these terms out is that the number of consistent Boolean networks may become super-exponential, and we may not be interested in finding all of them.

Previous learning algorithms for Boolean networks were based on exhaustive search and analyzed in Akutsu, Miyano, and Kuhara (1999). Their complexity was shown to be $O(2^{2^k} \cdot \binom{n}{k} \cdot n \cdot m \cdot \text{poly}(k))$ which is now considerably reduced by Eq. (7).

4.5. *Best-Fit Extension Problem, n variables*

We now show that a similar approach applies to the Best-Fit Extension Problem as well, when functions are from the class of all Boolean functions or the class of functions with k variables. Let us first concentrate on the former class and inspect the best matching function for a single gene only.

In the Best-Fit Extension Problem, as introduced in Section 3.2, we search for an extension $f \in \mathcal{C}$ of pdBf(T^*, F^*) that has minimum error size. In principle, we could solve the problem by computing the error size for all functions in $\mathcal{C}$. That would take time $O(2^{2^n} \cdot m \cdot \text{poly}(n))$, where 2^{2^n} is the number of functions in the class of all Boolean functions and $m \cdot \text{poly}(n)$ is the time needed to compute the error size for one function (see Eq. (3)). A faster solution can be found using a method similar to the one above.

Assume that we have two vectors $\mathbf{c}^{(0)}, \mathbf{c}^{(1)} \in \mathbb{R}^{2^n}$, where $\mathbb{R}$ denotes the set of real numbers, and $\mathbf{c}^{(0)}$ and $\mathbf{c}^{(1)}$ are indexed from 1 to 2^n and initially zero vectors. Then, during one pass over the given examples, $\mathbf{c}^{(0)}$ and $\mathbf{c}^{(1)}$ can be updated to

$$\begin{aligned} \mathbf{c}_i^{(0)} &= w(x), \quad \text{if } x \in F \wedge s(x) = i \\ \mathbf{c}_i^{(1)} &= w(x), \quad \text{if } x \in T \wedge s(x) = i \end{aligned}. \tag{8}$$

Elements of $\mathbf{c}_i^{(0)}$ and $\mathbf{c}_i^{(1)}$ that are not set in Eq. (8) remain zero-valued due to initialization. Let a fully determined candidate function be f with the corresponding truth table $\mathbf{f}$ having only binary elements. $\mathbf{f}_{s(x)}$ defines the output value for the input vector x as before. Let $\bar{\mathbf{f}}$

denote the complement of $\mathbf{f}$, i.e., a vector where all zeros and ones are flipped. Then, the error size of function f can be written as

$$\varepsilon(f) = \sum_{i=1}^{2^n} \mathbf{c}_i^{(\overline{\mathbf{f}_i})}. \tag{9}$$

It is now easy to see that error size is minimized when $\mathbf{c}_i^{(\overline{\mathbf{f}_i})}$ is minimum or, conversely, $\mathbf{c}_i^{(\mathbf{f}_i)}$ is maximum for all i. Thus, the truth table of the optimal Boolean function is $\mathbf{f}_i = \text{argmax}_j \mathbf{c}_i^{(j)}$. So, the truth table of a function minimizing Eqs. (3) and (9) can be found within a single pass of the set elements of the vectors $\mathbf{c}^{(0)}$ and $\mathbf{c}^{(1)}$. Therefore, when the time spent on memory initialization is ignored, the Best-Fit Extension Problem can be solved, for our class of functions, in time $O(m \cdot \text{poly}(n))$. Previously, a more general algorithm for solving the Best-Fit Problem for all transitive classes of Boolean functions was introduced in Boros, Ibaraki, and Makino (1998). The time complexity was shown to be $O((|T|+|F|)^3)$. Instead of finding a detailed algorithm for the class of all Boolean functions, the main objective of that study was to show that the problem is polynomial-time solvable for all transitive classes.

As introduced in Section 3.2, we may need to recompute the weights $w(x)$ in practice. However, instead of recomputing the weights in advance, it is better to embed the recomputation procedure into the above method. That can be done simply by changing Eq. (8) to

$$\begin{aligned} \mathbf{c}_i^{(0)} &= \sum_{x \in M\,:\,(s(x)=i)\wedge(x \in F)} w(x) \\ \mathbf{c}_i^{(1)} &= \sum_{x \in M\,:\,(s(x)=i)\wedge(x \in T)} w(x) \end{aligned}, \tag{10}$$

where M denotes the original multiset of measurements.

Example 2. Assume a simple hypothetical case where $k = 2$ and we are given sets T and F such that after applying Eq. (8) vectors $\mathbf{c}^{(0)}$ and $\mathbf{c}^{(1)}$ are updated to $\mathbf{c}^{(0)} = (4, 8, 1, 0)$ and $\mathbf{c}^{(1)} = (2, 2, 1, 5)$. Then the optimal function is found simply by using $\mathbf{f}_i = \text{argmax}_j \mathbf{c}_i^{(j)}$. Thus, $\mathbf{f}^{\text{opt}}$ is $(0, 0, 0, 1)$ and the corresponding error size is $\varepsilon(f^{\text{opt}}) = 5$. Note that when setting the third element of $\mathbf{f}^{\text{opt}}$, one needs to break a tie since both $\mathbf{c}_3^{(0)} = \mathbf{c}_3^{(1)} = 1$.

Gene expression time-series are usually noisy and the number of measurements is fairly small. Therefore, the best function is often not unique after applying the above inference scheme. Selecting only one predictor function per gene may lead to incorrect results. So, we may ultimately be interested in finding all functions f having error size smaller than or equal to some threshold, i.e., $\varepsilon(f) \leq \varepsilon_{\max}$. Vectors $\mathbf{c}^{(0)}$ and $\mathbf{c}^{(1)}$ can also be used for that purpose. For instance, the second best function, in view of the Best-Fit Extension Problem, is found by flipping only one element of the truth table of the best function f^{opt}. The flipped element, say the ith, has to be selected such that the corresponding absolute difference between elements $\mathbf{c}_i^{(0)}$ and $\mathbf{c}_i^{(1)}$ is minimum. This problem is discussed in Section 5.

4.6. Best-Fit Extension Problem, n genes and k variables

The generalization for n genes and k variables is as straightforward as already shown for the Consistency Problem in Sections 4.3 and 4.4. That is, the above method must be applied to all $\binom{n}{k}$ variable combinations and all n genes. So, the optimal solution of the Best-Fit Extension Problem for the entire Boolean network can be found in time

$$O(\tbinom{n}{k} \cdot n \cdot m \cdot \text{poly}(k)). \tag{11}$$

The problem of inferring gene regulatory networks under the Best-Fit extension paradigm was initially studied in Shmulevich et al. (2002b) and the proposed method was shown to be polynomial-time solvable for fixed k. In the case of varying k, the complexity of the algorithm in Shmulevich et al. (2002b) is $O(2^{2^k} \cdot \binom{n}{k} \cdot n \cdot m \cdot \text{poly}(k))$.

5. Finding set of functions with limited error-size

In this section we consider the problem of finding all functions having error size smaller than or equal to some given threshold $\varepsilon_{\max}$. In principle, we need to find a set of functions $\{f : \varepsilon(f) \leq \varepsilon_{\max}\}$. We first consider only a single function and single variable combination. Let us assume that we are considering the class of functions with k variables and have already found the optimal function, in the sense of the Best-Fit Extension Problem, using the method introduced above. Thus, we know vectors $\mathbf{c}^{(0)}$ and $\mathbf{c}^{(1)}$, error-size of the Best-Fit function $\varepsilon(f^{\text{opt}}) = \varepsilon_{\text{opt}}$, and the optimal binary function f^{opt} itself through its truth table $\mathbf{f}^{\text{opt}}$. In order to simplify the following computation/notation, we introduce some new variables. Define $\mathbf{c}$ to contain the absolute values of element-wise differences between $\mathbf{c}^{(0)}$ and $\mathbf{c}^{(1)}$, i.e., $\mathbf{c}_i = |\mathbf{c}_i^{(0)} - \mathbf{c}_i^{(1)}|$ and let $\mathbf{f}'$ denote truth table of a distorted (non-optimal) function f'. The truth table of f' can now be written as $\mathbf{f}' = \mathbf{f}^{\text{opt}} \oplus \mathbf{d}$, where $\mathbf{d} \in \{0, 1\}^{2^k}$ is the portion of distortion from the optimal function and $\oplus$ stands for addition modulo 2, i.e., $\mathbf{d} = \mathbf{f}' \oplus \mathbf{f}^{\text{opt}}$. Then, we get directly from Eq. (9) that

$$\varepsilon(f') = \sum_{i=1}^{2^k} \mathbf{c}_i^{(\overline{\mathbf{f}'_i})} = \sum_{i=1}^{2^k} \mathbf{c}_i^{(\overline{\mathbf{f}_i^{\text{opt}}})} + \sum_{i\,:\,\mathbf{d}_i=1} \mathbf{c}_i = \varepsilon_{\text{opt}} + \mathbf{c}^T\mathbf{d} \tag{12}$$

and can write the set of functions to be found in terms of truth tables as

$$\{\mathbf{f}^{\text{opt}} \oplus \mathbf{d} : \mathbf{c}^T\mathbf{d} \leq \varepsilon_{\max} - \varepsilon_{\text{opt}}\}. \tag{13}$$

At this stage the problem of finding the set of Boolean functions $\{f' : \varepsilon(f') \leq \varepsilon_{\max}\}$ has been converted to a problem of finding a subset $D \subset \{0, 1\}^{2^k}$ such that all elements $\mathbf{d} \in D$ share the property defined in Eq. (13). Perhaps the easiest way of continuing is as follows.

Let us first sort the vector $\mathbf{c}$ in increasing order and denote the sorted vector by $\mathbf{c}'$ and corresponding permutation (sorting) and inverse permutation operators by p and p^{-1}. Now,

we aim to find the set of distortion vectors D' in this new "permutation domain" that satisfy

$$\{\mathbf{d}' : \mathbf{c}'^T\mathbf{d}' \leq \varepsilon_{\max} - \varepsilon_{\text{opt}}\}. \tag{14}$$

Note that this is closely related to Eq. (13). The following greedy recursive algorithm can be used to find D' such that their inverse permutations together with $\mathbf{f}^{\text{opt}}$ define all functions f, satisfying $\varepsilon(f) \leq \varepsilon_{\max}$. The operation of the algorithm can be described as follows. Conceptually, the algorithm builds a tree such that each node in the tree corresponds to one acceptable permuted distortion vector. (A tree is not built in practice, but helps to understand the operation of the algorithm.) At the depth 0, which corresponds to the root of the tree, it has permuted distortion vector whose Hamming weight is zero, i.e., the zero vector $(0 \ldots 0)$. At the depth 1, the tree has all permuted distortion vectors whose Hamming weight is 1 and they satisfy Eq. (14) and so on for deeper levels. In detail, on level 1 we have nodes corresponding to vectors $(100 \ldots 0), (010 \ldots 0), \ldots, (000 \ldots 010 \ldots 0)$, where the index i of the 1 element in the last vector is such that next position $i + 1$ in the sorted vector $\mathbf{c}'$ would alone force the error to be larger than the given threshold. That is, $(000 \ldots 010 \ldots 0)\mathbf{c}' \leq \varepsilon_{\max} - \varepsilon_{\text{opt}}$ holds for the last vector in the previous list, but would not hold for the "next" vector. Below the node (vector) $(10 \ldots 0)$ at the level 1, we have nodes (vectors) at the level 2 $(11000 \ldots 0), (10100 \ldots 0), (10010 \ldots 0), \ldots, (10 \ldots 010 \ldots 0)$, where the index i of the right most 1 element in the last vector in the previous list is such that $(10 \ldots 010 \ldots 0)\mathbf{c}' \leq \varepsilon_{\max} - \varepsilon_{\text{opt}}$ holds for that vector but would not hold for the "next" vector (the one where the right most 1 is moved one position to the right), and so on for the remaining nodes. The algorithm itself is shown in figure 1. In the following, we assume that D' and $\mathbf{c}'$ are available and the same at all recursion levels.

Statement for the while-loop is assumed to be evaluated in "short-circuit evaluation" fashion that prevents indexing $\mathbf{c}'$ over its length. When we initialize $\mathbf{d}' \leftarrow (0 \ldots 0)$, $D' \leftarrow \mathbf{d}'$ and $i_0 \leftarrow 0$, the set D' will contain proper "distortion" vectors, or their p permutations, after the function call `Distortion-Vectors`$(\mathbf{d}', \varepsilon_{\text{opt}}, i_0)$, assuming $\varepsilon(f^{\text{opt}}) \leq \varepsilon_{\max}$. In order to find the final distortion vectors we still need to apply the inverse permutation p^{-1} to the found vectors in D'. Then, truth tables of the searched functions, satisfying $\{f : \varepsilon(f) \leq \varepsilon_{\max}\}$, can be formed as $\{\mathbf{f}^{\text{opt}} \oplus \mathbf{d} : \mathbf{d} \in p^{-1}(D')\}$.

Example 3. We continue our previous Example 2 in order to illustrate the operation of the above algorithm. Using the values for $\mathbf{c}^{(0)}$ and $\mathbf{c}^{(1)}$ shown in Example 2 we get

```
1 Distortion-Vectors(d', ε, i)
2 j ← i + 1
3 while j ≤ 2^k ∧ ε + c'_j ≤ ε_max
4   d'_j ← 1
5   D' ← D' ∪ {d'}
6   Distortion-Vectors(d', ε + c'_j, j)
7   d'_j ← 0
8   j ← j + 1
9 endwhile
```

Figure 1. A greedy algorithm that can be used to find the set of functions $\{f : \varepsilon(f) \leq \varepsilon_{\max}\}$.

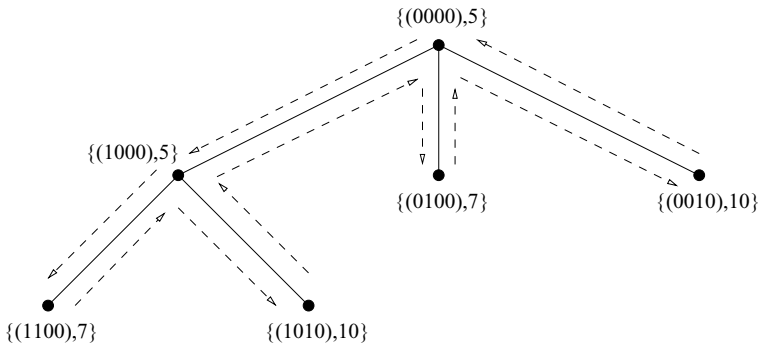


Figure 2. The (conceptual) tree build by the `Distortion-Vectors`-algorithm. Each node is associated with an ordered pair consisting of a permuted distortion vector and corresponding error size computed using Eqs. (12) and (14). All distortion vectors shown in figure 2 are added into D'. Dashed arrows show the running (construction) order of the nodes where the root is the starting node.

$\mathbf{c} = (2, 6, 0, 5)$, and further $\mathbf{c}' = (0, 2, 5, 6)$ after sorting $\mathbf{c}$. Permutation (sorting) operation is defined by the index vector (3, 1, 4, 2) which defines the original positions of the elements in $\mathbf{c}'$. We assume $\varepsilon_{\max}$ to be 10 and therefore $\varepsilon_{\max} - \varepsilon_{\text{opt}} = 5$. Then we are ready to use the `Distortion-Vectors`-algorithm. The operation of the algorithm using the conceptual tree is shown in figure 2. Each node is associated with an ordered pair consisting of a permuted distortion vector and corresponding error size computed using Eqs. (12) and (14). Precisely all distortion vectors shown in figure 2 are added into D', thus $D' = \{(0000), (1000), (1100), (1010), (0100), (0010)\}$. Dashed arrows show the running (construction) order of the nodes where the root is the starting node.

Final distortion vectors are found by applying the inverse permutation operator p^{-1} to D', i.e., $p^{-1}(D') = \{(0000), (0010), (1010), (0011), (1000), (0001)\}$. Finally, we get the set of functions $\{f : \varepsilon(f) \leq \varepsilon_{\max}\} = \{\mathbf{f}^{\text{opt}} \oplus \mathbf{d} : \mathbf{d} \in p^{-1}(D')\} = \{(0001), (0011), (1011), (0010), (1001), (0000)\}$.

Without going into details, we give a sketch of the proof of the correctness of the above algorithm, also relying on the conceptual tree interpretation. The first observation is that each node is associated with the correct error size. Clearly, this holds for the root node, assuming the algorithm is originally called with error size ε_{opt}. Then, we can inductively validate the first claim for deeper levels. Because each node is associated with the correct error size it follows that only proper permuted distortion vectors are added into D'. Utilizing the increasing order of $\mathbf{c}'$, similar inductive observations over the depth of the nodes can be made to prove that all proper permuted distortion vectors are added into D'. So, in total we have that the set of permuted distortion vectors found by the above algorithm is equal to that in Eq. (14).

The time complexity of the algorithm is problem dependent. That is, it depends heavily on the number of functions in the "true" set $\{f : \varepsilon(f) \leq \varepsilon_{\max}\}$. However, it is fairly obvious that proper permuted distortion vectors are added into D' only once since nodes in separate branches of the tree are associated with vectors (binary strings) having different prefixes. So,

the above procedure makes no fruitless job in that sense. We can also consider the number of "unnecessary" checks in the while-loop (the third row of the algorithm) here, although this measure is also problem dependent. Because the while-loop is broken after the first failure we see that the number of "unnecessary" checks is upper-bounded by the number of nodes in the tree which further equals the number of proper functions. So, the computational complexity, when including the sorting step, can be written as $O(2^k \cdot \log(2^k) \cdot \text{poly}(k) + |\{f : \varepsilon(f) \leq \varepsilon_{\max}\}|) = O(2^k \cdot k \cdot \text{poly}(k) + |\{f : \varepsilon(f) \leq \varepsilon_{\max}\}|)$ assuming that we are considering the class of functions with k variables.

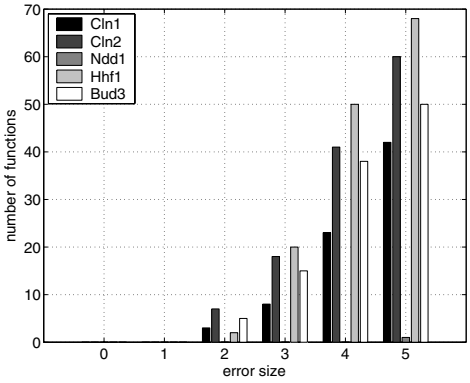
In order to find the functions for the whole Boolean network one needs to apply the above procedure for all $\binom{n}{k}$ variable combinations and n genes essentially multiplying the time complexity by $\binom{n}{k} \cdot n$.

6. Experiments

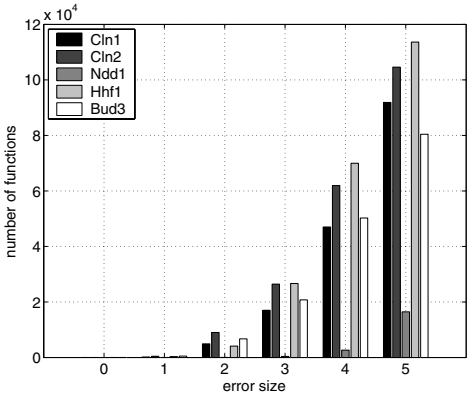
In this section we present inference results for the cdc15 yeast gene expression time-series data set taken from Spellman et al. (1998) where 799 cell cycle regulated genes were identified. The cell cycle regulated genes were split into five groups in each of which genes behaved similarly. The groups were referred to as $G1$, S, $G2$, M and $M/G1$, corresponding to different states of the cell cycle. In order to limit the computational complexity, we restricted our experiments only to those 799 cell cycle regulated genes. Further, we did not infer the entire regulatory network, but searched the functions for a selected set of genes. In this experiment, we were interested in the set of five genes: {Cln1, Cln2, Ndd1, Hhf1, Bud3}. See Spellman et al. (1998) for information about function, phase of peak expression, and other details of the selected genes.) Instead of selecting only one function per gene, we searched all functions with the error size not exceeding a pre-determined threshold $\varepsilon_{\max} = 5$. (When the number of variables is 3 we used $\varepsilon_{\max} = 3$ because the number of functions is quite large for $\varepsilon = 4, 5$.)

We preprocessed the expression data using the following steps. The normalization was carried out as in Spellman et al. (1998). The missing expression values in the cdc15 experiment were estimated by using the weighted K-nearest neighbors method introduced in Troyanskaya et al. (2001). The number of neighbors in the estimation method was set to 15 and the Euclidean norm was used as the basis for the similarity measure, as suggested in Troyanskaya et al. (2001). We omitted the genes that have more than 20% missing values. In order to check the biological relevance of our results we added some genes that are known to regulate the genes of interest. Some of the added genes were not found to be cell-cycle regulated in Spellman et al. (1998). As a consequence, the set of 799 cell cycle regulated genes was reduced to 733 genes. The data binarization was performed using the quantization method described in Shmulevich and Zhang (2002c) together with a global threshold equal to 0.

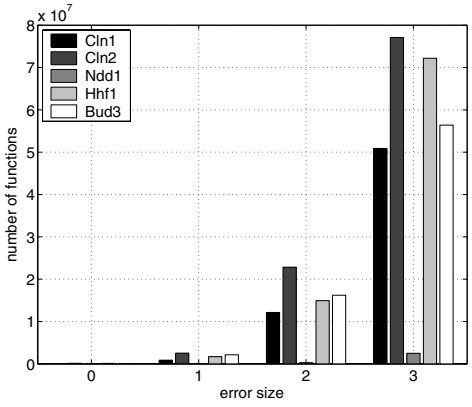
Since no measures of quality are available for our set of measurements, we used unity weight, $w(x) = 1$, for all measurements. The maximum indegree of the functions, k, was set to 1, 2, and 3. As a result, the histograms of the error size of the found functions for all the selected genes are shown in figure 3. The horizontal axis corresponds to the error size and



(a)



(b)



(c)

Figure 3. The histograms of the error size of the found functions for all the selected genes. The maximum indegree is (a) $k = 1$, (b) $k = 2$ and (c) $k = 3$. The bars for the different genes are coded as shown on the graphs.

Table 1. The number of functions having error size between 0 and 5 for all the selected genes. The maximum indegree is set to $k = 1$, $k = 2$ and $k = 3$. For indegree $k = 3$ we only show the number of functions for error sizes 0 through 3.

Indegree	Gene\Error size	0	1	2	3	4	5
$k = 1$	Cln1	0	0	3	8	23	42
	Cln2	0	0	7	18	41	60
	Ndd1	0	0	0	0	0	1
	Hhf1	0	0	2	20	50	68
	Bud3	0	0	5	15	38	50
$k = 2$	Cln1	1	175	4930	17022	47018	91895
	Cln2	12	432	9018	26425	61933	104595
	Ndd1	0	1	34	378	2664	16434
	Hhf1	21	332	4110	26665	69989	113637
	Bud3	16	510	6712	20769	50257	80427
$k = 3$	Cln1	27051	860663	12131506	50893601		
	Cln2	118042	2526975	22822482	77088111		
	Ndd1	420	13657	246522	2495556		
	Hhf1	95834	1696500	14920744	72192691		
	Bud3	93683	2127964	16209816	56397855		

the vertical axis shows the number of functions for each gene that have the corresponding error size. The error sizes are also shown in numerical form in Table 1. Note that the set of functions with maximum indegree $k = i + 1$ contains all the functions with maximum indegree $k = i$. Recall that the column corresponding to zero error size contains the numbers of consistent functions.

Recently, yeast cell cycle transcriptional activators have been studied by Simon et al. (2001) by combining both expression measurements and location analysis. In that study, of particular interest were certain transcriptional regulators and cyclin genes, including the genes Cln1, Cln2, Ndd1 and Hhf1. We compare our results with the ones shown in that paper in order to validate the biological significance of our results. We also want to emphasize that, in order to simplify the function inference task, we could have concentrated only on the genes that are known to regulate each other. That is not, however, the purpose of the proposed inference method. Instead, we really want to search through the whole "space" of models (functions). The key motivation for doing so is to search new possible predictor functions not already known.

In the study by Simon et al. (2001), the complex Swi4/Swi6 was found to be a transcriptional regulator of the cyclin gene Cln1. In our analysis, the Best-Fit predictor function for the gene Cln1, with input variables {Swi4,Swi6}, had error size 3. Recently, genome-wide location analysis has also revealed that genes such as Mbp1 and Ndd1 can bind to the gene Cln1 (Lee et al., 2002). So, since the complex Swi4/Swi6 and genes Mbp1 and Ndd1 all can presumably regulate the gene Cln1, it is instructive to look at the three variable predictor

functions having input variables {Swi4,Swi6,Mbp1} and {Swi4,Swi6,Ndd1}. The Best-Fit predictor function for those input variables turned out to have error size 2. Even though the error size for those functions is not the smallest, it is in clear agreement with the results shown in Simon et al. (2001) and Lee et al. (2002). For instance, the two variable predictor {Swi4,Swi6} is among the top 0.5% of all two variable predictors.[3] Similarly, the three variable predictors {Swi4,Swi6,Mbp1} and {Swi4,Swi6,Ndd1} are both among the top 0.08% of all three variable predictors. For the gene Cln2, a known regulator is Rme1 (Banerjee & Zhang, 2002). The Best-Fit error size for that single variable function is 4, and it is found to be among the top 2.3% of all one variable predictors. The transcriptional regulator Swi4/Swi6 has also been found to be able to regulate the gene Cln2 (Simon et al., 2001). Based on the used data, the Best-Fit method did not find that variable combination alone as a potential regulator (error size is 5). However, when looking at the three variable functions, e.g. with variables {Swi4,Swi6,Rme1} and {Swi4,Swi6,Clb2}, (Clb2 can also regulate the gene Cln2 (Simon et al., 2001)) the minimum error sizes were found to be as low as 3 and 2, respectively. Previous studies have shown that complexes Swi4/Swi6 and Swi6/Mbp1 can bind to the promoter region of the gene Ndd1. So, one would expect a function with variables {Swi4,Swi6,Mbp1} to be a reasonably good predictor for the gene Ndd1. However, the binary Best-Fit analysis on the used data set did not give very strong support for these genes to be a probable good predictor set for the gene Ndd1; error size was found to be 5. One possible reason, in addition to noise, could be that the expression profile of the Ndd1 has 4 missing measurements in the cdc15 experiment. The interpolated values may have caused some artificial effects.

In general, our results agree with most of the known regulatory behavior ((Simon et al., 2001; Banerjee & Zhang, 2002; Lee et al., 2002). The Best-Fit functions for the known transcriptional regulators are among the best ones in almost every case. However, we can also identify some problems in our experiments. The first is the insufficient length of the time-series or, alternatively, the large size of the space of possible predictors. That is, the probability of finding a good predictor by chance is fairly high. For instance, since we are looking at cell cycle regulated genes only, some two genes A and B may simply have similar (periodic) expression profiles with a proper "phase difference." If gene A has peaks and troughs in its expression profile slightly before gene B, it may be found to regulate gene B. In that kind of case, we are not able to decide whether or not the found transcriptional factor is spurious. Further biological measurements such as location data (Ren et al. 2000) or functional studies may be needed. Second, as the Best-Fit framework incorporates positive weights, it would have been beneficial if one had been able to utilize them in this experiment. We, on the other hand, only used equal weights for each vector in T and F (see Section 7 for further discussion).

7. Discussion

The current gene expression measurements contain a considerable amount of noise and "low quality" measurements are usually expunged from the subsequent analysis. Alternatively, each measurement could be associated with some indicative measure of its quality. Then, instead of throwing away the low quality measurements, one is able to control their relative

influence in the following data analysis steps by down-weighting the most unreliable ones. Estimating the qualities for gene expression measurements has been discussed, e.g. by Chen et al. (2002) and Smyth, Yang, and Speed (2003) quite recently. Ones those quality measurements become available, they can be directly used in the gene regulatory network inference in the Best-Fit Extension framework.

As in most estimation tasks, over-fitting is an issue in inference of gene regulatory networks. This is particularly evident from our experiments discussed in the previous section. That is, as maximum indegree k increases, we get a better fitting for the training set, resulting in overtrained network models. A known fact is that models that are "too adapted" to the training set loose their generalization capability for the cases outside the training set. In order to avoid possible over-fitting, it may be useful to constrain the search of possible predictor functions only to a certain (restricted) class of functions, such as canalizing functions, transitive classes, etc. Unfortunately, the inference methods introduced in this paper do not directly apply to all classes of Boolean functions. This is because the found optimal functions are not guaranteed to belong to any restricted function class other than the one that contains all k-variable functions. However, the availability of efficient generation and membership testing algorithms for special classes of Boolean functions will alleviate this problem. Nonetheless, constraining the search space without strong prior knowledge may be risky.

Another possible approach to prevent over-fitting is by utilizing the information-theoretic Minimum Description Length (MDL) principle. The use of the MDL principle in gene regulatory network learning was introduced by Tabus and Astola (2001), who considered the MDL criterion-based two-part code together with a particular encoding scheme for finding good predictor functions for a single gene. Although their approach was introduced with ternary data, it is also straightforward to apply it in a binary setting. In detail, the two-part code consists of the code length for the measured data given the model and the code length for the model itself, often denoted by $\mathcal{L}(\mathcal{D}|\mathcal{M})$ and $\mathcal{L}(\mathcal{M})$. The encoding method in Tabus and Astola (2001) encodes only the model residuals, i.e., the errors generated by the model (function) of interest. It is easily observed that the Best-Fit Extension method can be utilized in the MDL-based inference approach as well. That is, with unity weights $w(x) = 1$ for all $x \in T \cup F$, the error-size of a Boolean function in the Best-Fit method is just the number of misclassifications. That measure provides a fast way of computing the MDL criterion in gene regulatory network inference. For more details, see Tabus and Astola (2001) and the references therein.

One drawback of Boolean networks is that they are inherently deterministic. Shmulevich et al. (2002a) recently introduced an extension of the standard Boolean networks, called Probabilistic Boolean Networks (PBN). This model relaxes the inherent determinism of Boolean networks, but still provides a rule-based approach to modeling. In principle, uncertainty is brought into the model as follows. Instead of only one predictor function f_i a set of predictors $F_i = \{f_1^{(i)}, \ldots, f_{l(i)}^{(i)}\}$ together with the corresponding selection probabilities $C_i = \{c_1^{(i)}, \ldots, c_{l(i)}^{(i)}\}$ are assigned to the ith gene. Then, during each updating step of the network, each gene x_i can randomly take its predictor function, according to the distribution C_i, from the corresponding set of functions F_i.[4] As introduced by Shmulevich et al. (2002a), inference of PBNs can be carried out by employing the coefficient of determination

(COD) (Dougherty, Kim, & Chen, 2000) in selection of the predictors for a given gene. The predictors themselves can be found, for example, by applying the standard Boolean network inference methods, such as the proposed methods for the Consistency and the Best-Fit Extension Problem, or the MDL-based inference. Thus, the need to find sets of "good" functions in the PBN setting is another reason why we were interested in finding more than one candidate function per gene in Section 5. The relationship between the PBN and the (dynamic) Bayesian network was discussed in Shmulevich et al. (2002a). For instance, it was shown that the conditional probability of a gene (node in PBN) given its (causal) parents can be expressed in terms of the Boolean functions and selection probabilities. As noted by Shmulevich et al. (2002a), such a formulation may open up the possibility of selecting models using standard Bayesian learning techniques. In view of model complexity considerations, this is particularly interesting since the Bayesian model selection criteria have an automatic built-in *Occam's razor* principle (see e.g. MacKay, 1992).

Notes

1. Although poly(n) will be equal to n in most cases, we will use poly(n) for generality.
2. Repeated measurements can always be combined into single measurements without changing the result of the Consistency Problem.
3. Note that the number of all possible 2-variable predictors is already $2^{2^2} \cdot \binom{733}{k} = 4292448$. For three variables the number of all predictors is as large as $2^{2^3} \cdot \binom{733}{k} = 16734823936$.
4. Strictly speaking, this holds only for so-called independent PBNs. There also exist dependent PBNs.

References

Akutsu, T., Kuhara, S., Maruyama, O., & Miyano, S. (1998). Identification of gene regulatory networks by strategic gene disruptions and gene overexpressions. In *Proc. the 9th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'98)*, (pp. 695–702).

Akutsu, T., Miyano, S., & Kuhara, S. (1999). Identification of genetic networks from a small number of gene expression patterns under the Boolean network model. *Pacific Symposium on Biocomputing, 4*, 17–28.

Akutsu, T., Miyano, S., & Kuhara, S. (2000). Inferring qualitative relations in genetic networks and metabolic pathways. *Bioinformatics, 16*, 727–734.

Arnone, M. I., & Davidson, E. H. (1997). The hardwiring of development: Organization and function of genomic regulatory systems. *Development, 124*, 1851–1864.

Banerjee, N., & Zhang, M. Q. (2002). Functional genomics as applied to mapping transcription regulatory networks. *Current Opinion in Microbiology, 5:3*, 313–317.

Boros, E., Ibaraki, T., & Makino, K. (1998). Error-Free and Best-Fit Extensions of partially defined Boolean functions. *Information and Computation, 140*, 254–283.

Chen, T., He, H. L., & Church, G. M. (1999). Modeling gene expression with differential equations. *Pacific Symposium on Biocomputing, 4*, 29–40.

Chen, Y., Dougherty, E., & Bittner, M. (1997). Ratio-based decisions and the quantitative analysis of cDNA microarray images. *Journal of Biomedical Optics, 2*, 364–374.

Chen, Y., Kamat, V., Dougherty, E. R., Bittner, M. L., Meltzer, P. S., & Trent, J. M. (2002). Ratio statistics of gene expression levels and applications to microarray data analysis. *Bioinformatics, 18*, 1207–1215.

Cormen, T. H., Leiserson, C. E., & Rivest, R. L. (1998). *Introduction to Algorithms*. MIT Press.

D'Haeseleer, P., Wen, X., Fuhrman, S., & Somogyi, R. (1999). Linear modeling of mRNA expression levels during CNS development and injury. *Pacific Symposium on Biocomputing, 4*, 41–52.

de Jong, H. (2002). Modeling and simulation of genetic regulatory systems: A literature review. *Journal of Computational Biology, 9:1*, 67–103.

Dougherty, E. R., Kim, S., & Chen, Y. (2000). Coefficient of determination in nonlinear signal processing. *Signal Processing, 80:10*, 2219–2235.

Friedman, N., Linial, M., Nachman, I., & Pe'er, D. (2000). Using Bayesian networks to analyze expression data. *Journal of Computational Biology, 7*, 601–620.

Glass, L., & Kauffman, S. A. (1973). The logical analysis of continuous non-linear biochemical control networks. *Journal of Theoretical Biology, 39*, 103–129.

Hartemink, A., Gifford, D., Jaakkola, T., & Young, R. (2001). Using graphical models and genomic expression data to statistically validate models of genetic regulatory networks. *Pacific Symposium on Biocomputing, 6*, 422–433.

Hasty, J., McMillen, D., Isaacs, F., & Collins, J. J. (2001). Computational studies of gene regulatory networks: In numero molecular biology. *Nature Reviews Genetics, 2*, 268–279.

Huang, S. (1999). Gene expression profiling, genetic networks and cellular states: An integrating concept for tumorigenesis and drug discovery. *Journal of Molecular Medicine, 77*, 469–480.

Ideker, T. E., Thorsson, V., & Karp, R. M. (2000). Discovery of regulatory interactions through perturbation: Inference and experimental design. *Pacific Symposium on Biocomputing, 5*, 302–313.

Karp, R. M., Stoughton, R., & Yeung, K. Y. (1999). Algorithms for choosing differential gene expression experiments. *RECOMB99* (pp. 208–217). ACM.

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

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

Kerr, M. K., Leiter, E. H., Picard, L., & Churchill, G. A. (2002). Sources of variation in microarray experiments. In W. Zhang, & I. Shmulevich (Eds.), *Computational and Statistical Approaches to Genomics*. Boston: Kluwer Academic Publishers.

Lee, T. I., Rinaldi, N. J., Robert, F., Odom, D. T., Bar-Joseph, Z., Gerber, G. K., Hannett, N. M., Harbison, C. T., Thompson, C. M., Simon, I. et al. (2002). Transcriptional regulatory networks in *Saccharomyces cerevisiae*. *Science, 298*, 799–804.

Liang, S., Fuhrman, S., & Somogyi, R. (1998). REVEAL, A general reverse engineering algorithm for inference of genetic network architectures. *Pacific Symposium on Biocomputing, 3*, 18–29.

Maki, Y., Tominaga, D., Okamoto, M., Watanabe, S., & Eguchi, Y. (2001). Development of a system for the inference of large scale genetic networks. *Pacific Symposium on Biocomputing, 6*, 446–458.

MacKay, D. J. C. (1992). Bayesian interpolation. *Neural Computation, 4:3*, 415–447.

McAdams, H. H., & Arkin, A. (1997). Stochastic mechanisms in gene expression. *Proc. Natl. Acad. Sci. USA, 94*, 814–819.

McAdams, H. H., & Arkin, A. (1999). It's a noisy business! Genetic regulation at the nanomolar scale. *Trends in Genetics, 15*, 65–69.

Mestl, T., Plahte, E., & Omholt, S. W. (1995). A mathematical framework for describing and analyzing gene regulatory networks. *Journal of Theoretical Biology, 176*, 291–300.

Murphy, K., & Mian, S. (1999). Modelling gene expression data using dynamic Bayesian networks. Technical Report, University of California, Berkeley.

Noda, K., Shinohara, A., Takeda, M., Matsumoto, S., Miyano, S., & Kuhara, S. (1998). Finding genetic network from experiments by weighted network model. *Genome Informatics, 9*, 141–150.

Ren, B., Robert, F., Wyrick, J. J., Aparicio. O., Jennings, E. G., Simon, I., Zeitlinger, J., Schreiber, J., Hannett, N., Kanin, E. et al. (2000). Genome-wide location and function of DNA binding proteins. *Science, 290*, 2306–2309.

Shmulevich, I., Dougherty, E. R., Seungchan, K., & Zhang, W. (2002a). Probabilistic Boolean networks: A rule-based uncertainty model for gene regulatory networks. *Bioinformatics, 18*, 261–274.

Shmulevich, I., Saarinen, A., Yli-Harja, O., & Astola, J. (2002b). Inference of genetic regulatory networks under the Best-Fit Extension paradigm. In W. Zhang, and I. Shmulevich (Eds.), *Computational And Statistical Approaches To Genomics*. Boston: Kluwer Academic Publishers.

Shmulevich, I., & Zhang, W. (2002c). Binary analysis and optimization-based normalization of gene expression data. *Bioinformatics, 18*, 555–565.

Simon, I., Barnett, J., Hannett, N., Harbison, C. T., Rinaldi, N. J., Volkert, T. L., Wyrick, J. J., Zeitlinger, J., Gifford, D. K., Jaakkola, T. S., & Young, R. A. (2001). Serial regulation of transcriptional regulators in the yeast cell cycle. *Cell, 106*, 697–708.

Smolen, P., Baxter, D. A., & Byrne, J. H. (2000). Mathematical modeling of gene networks. *Neuron, 26*, 567–580.

Smyth, G. K., Yang, Y. H., & Speed, T. (2003). Statistical issues in cDNA microarray data analysis. In M. J. Brownstein, & A. B. Khodursky (Eds.), *Functional Genomics: Methods and Protocols, Methods in Molecular Biology series* (pp. 111–136). Totowa, NJ: Humana Press. To appear.

Somogyi, R., & Sniegoski, C. (1996). Modeling the complexity of gene networks: Understanding multigenic and pleiotropic regulation. *Complexity, 1*, 45–63.

Spellman, P. T., Sherlock, G., Zhang, M. Q., Iyer, V. R., Anders, K., Eisen, M. B., Brown, P. O., Botstein, D., & Futcher, B. (1998). Comprehensive identification of cell cycle-regulated genes of the yeast *Saccharomyces cerevisiae* by Microarray Hybridization. *Molecular Biology of the Cell, 9*, 3273–3297.

Tabus, I., & Astola, J. (2001). On the use of MDL principle in gene expression prediction. *Journal of Applied Signal Processing, 4*, 297–303.

Tabus, I., Rissanen, J., & Astola, J. (2002). Normalized maximum likelihood models for Boolean regression with application to prediction and classification in genomics. In W. Zhang, & I. Shmulevich (Eds.), *Computational And Statistical Approaches To Genomics*. Boston: Kluwer Acadmic Publishers.

Thieffry, D., Huerta, A. M., Pèrez-Rueda, E., & Collado-Vides, J. (1998). From specific gene regulation to genomic networks: A global analysis of transcriptional regulation in *Escherichia coli*. *BioEssays, 20*, 433–440.

Troyanskaya, O., Cantor, M., Sherlock, G., Brown, P., Hastie, T., Tibshirani, R., Botstein, D., & Altman, R. B. (2001). Missing value estimation methods for DNA microarrays. *Bioinformatics, 17*, 520–525.

Vohradsky, J. (2001). Neural model of the genetic network. *The Journal of Biological Chemistry, 276:39*, 36168–36173.

Weaver, D. C., Workman, C. T., & Stormo, G. D. (1999). Modeling regulatory networks with weight matrices. *Pacific Symposium on Biocomputing, 4*, 112–123.

Yli-Harja, O., Linne, M.-L., & Astola, J. (2001). On the use of cDNA microarray data in Boolean network inference. In *Proc. Conf. on Computer Science and Information Technologies* (pp. 405–409). Yerevan, Armenia.

Received June 18, 2002
Revised November 7, 2002
Accepted November 8, 2002
Final manuscript January 14, 2003