Mach Learn (2006) 65:229–245
DOI 10.1007/s10994-006-9014-z

# An efficient top-down search algorithm for learning Boolean networks of gene expression

**Dougu Nam · Seunghyun Seo · Sangsoo Kim**

Received: 8 March 2004 / Revised: 1 September 2005 / Accepted: 24 April 2006 / Published online: 30 June 2006
Springer Science + Business Media, LLC 2006

**Abstract** Boolean networks provide a simple and intuitive model for gene regulatory networks, but a critical defect is the time required to learn the networks. In recent years, efficient network search algorithms have been developed for a noise-free case and for a limited function class. In general, the conventional algorithm has the high time complexity of $O(2^{2^k} mn^{k+1})$ where $m$ is the number of measurements, $n$ is the number of nodes (genes), and $k$ is the number of input parents. Here, we suggest a simple and new approach to Boolean networks, and provide a randomized network search algorithm with average time complexity $O(mn^{k+1}/(\log m)^{(k-1)})$. We show the efficiency of our algorithm via computational experiments, and present optimal parameters. Additionally, we provide tests for yeast expression data.

**Keywords** Boolean network · Data consistency · Random superset selection · Core search · Coupon collection problem

## 1. Introduction

DNA microarray technologies provide information on genome-wide expression under various genetic, chemical, and environmental perturbations. In the past decade, a number of

**Editor:** David Page

D. Nam (✉)
National Genome Information Center, Korea Research Institute of Bioscience and Biotechnology, Eoun-dong 52, Yusong-gu, Daejeon 305-333, Rep. of Korea
e-mail: dunam@kribb.re.kr

S. Seo
Department of Mathematics, Seoul National University, Shinlim-dong, Kwanak-gu, Seoul 151-747, Rep. of Korea
e-mail: shseo@snu.ac.kr

S. Kim
National Genome Information Center, Korea Research Institute of Bioscience and Biotechnology, Eoun-dong 52, Yusong-gu, Daejeon 305-333, Rep. of Korea; Department of Bioinformatics, Soongsil Univ., Sangdo 5-Dong, Dongjak-Gu, Seoul 156-743, Rep. of Korea
e-mail: sskimb@ssu.ac.kr

computational algorithms have been developed to better understand expression data. The widely used clustering analyses have provided useful biological information by identifying genes with similar expression patterns. However, gene expression is determined by complex and combinatorial activities of genes, proteins, and metabolites, and such ‘causal’ or directional regulatory relationships may not be simply described by similarity measures. The first attempt to model such regulatory relationships among genes was the Boolean network model (Kauffman et al., 1967). This model quantizes gene expression values into two discrete levels ‘on’ and ‘off’, and aims to describe deterministic logical regulatory relationships of gene expression. Other studies (Liang et al., 1998; Akutsu et al., 1999), show how Boolean regulatory networks can be learned from large scale expression data. Subsequently, Bayesian networks (Friedman et al., 2000) and Probabilistic Boolean networks (Shmulevich et al., 2002) were suggested to model the probabilistic gene regulations.

Although Boolean networks provide a simple and intuitive model for regulatory networks, the time required to learn the networks is limiting. The conventional exhaustive search algorithm Akutsu et al. (1999a) has the high time complexity of $O(2^{2^k} mn^{k+1})$ where $m$ is the number of measurements, $n$ is the number of genes, and $k$ is the number of input parents. An efficient $O(2^{2^k} mn^k)$ algorithm was suggested using trie structure for the noise-free case in Akutsu et al. (1999b), and an even more efficient greedy algorithm was developed for AND-OR function classes in previous studies (Akutsu et al., 2003; Fukagawa and Akutsu, 2003). However, real data contain much noise and AND-OR functions may not be sufficient to represent the genetic information. For noisy data and the whole function class, an important gain of $2^{2^k}$ was obtained in Lähdesmäki et al. (2003).

We propose a fast, randomized algorithm with average time complexity $O(mn^{k+1}/(\log m)^{(k-1)})$. The key idea was derived from the fact that if some input variables have a consistent functional relationship with an output variable, any superset of the input variables will have a consistent functional relationship with the output variable. Because the average-case time complexities of the previous algorithms are the same as the worst-case time complexities for noisy data, the proposed algorithm provides a substantial improvement. The performance of the algorithm presented here can be more or less worse than the test results presented in this article if the parameters, e.g. the superset size (see Section 3), are not chosen optimally. However, we can estimate appropriate parameter values by comparing the computing times for sampled nodes before learning the Boolean networks of all nodes.

In Section 2, we show a simplified approach to Boolean networks and formulate the best-fit extension problem (Boros et al., 1998). We show that the gain of $2^{2^k}$ can be simply obtained by our approach. Hence, the algorithm suggested in Section 2 is equivalent to that of Lähdesmäki et al. (2003). In Section 3, we describe and analyze the proposed algorithm ‘top-down search algorithm (TDSA)’ and show that TDSA has the additional gain of $(\log m)^{(k-1)}$. In Section 4, we give computational experiments showing that TDSA is by and large from dozens to hundreds times faster than a recent algorithm given by Lähdesmäki et al. (2003) for relatively low level of noise ($5\% \sim 10\%$ inconsistent mapping). Lastly, we test TDSA for yeast expression data where the ‘inconsistent’ mapping ratio reached $\sim 20\%$. Even in this highly noisy data, TDSA found multiple parent tuples for each gene quickly and accurately for given thresholds of mapping consistency.

## 2. A simplified approach to Boolean networks

In this article, we only consider two expression levels ‘0’ and ‘1’ for simplicity. The data we use are the quantized expression profiles, an $n \times m$ matrix.

**Table 1** Inconsistent data

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Input | $g_{i_1}$ | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| | $g_{i_2}$ | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 |
| Output | $g_i$ | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |

**Fig. 1** A tower shape diagram

$$\begin{array}{ccccc}
 & & \boxed{1} & & \\
 & & 0 & & 0 \\
 & 1 & 0 & 1 & 0 \\
 & 1 & 0 & 1 & 0 \\
\hline
(g_{i_1}, g_{i_2}): & (0,0) & (0,1) & (1,0) & (1,1)
\end{array}$$

## 2.1. Noise-free Boolean networks

The task of constructing Boolean networks from data is to find the parent nodes (genes) for each node that give ‘consistent’ mapping rules through the $m$ experiments between the expression profiles of candidate parents (input) and each gene’s profile (output). We call the tuple of input values and the output value $\langle g_{i_1}(j), \ldots, g_{i_k}(j); g_i(j)\rangle$ an *example* for each $j = 1, 2, \ldots, m$.

To illustrate the ‘consistent’ mapping, we assume $k = 2$ for gene $g_i$ and assume $(g_{i_1}, g_{i_2})$ is one candidate parent tuple for $g_i$. Suppose we have the following expression profiles for $(g_{i_1}, g_{i_2})$ and $g_i$ as in Table 1. These data can be graphically represented by tower shapes as in Fig. 1. The output values are accumulated vertically on each possible tuple of input values. The former three output values for the input value (0,1) were 0, but in the 11th example, the output value is 1 for the same input value, where the consistency is broken: the boxed number in Fig. 1. The number of towers will be doubled whenever one more input parent is added. If the output values are the same for each possible input value, we say the parent tuple $(g_{i_1}, \ldots, g_{i_k})$ is consistent with $g_i$ and we call $\langle (g_{i_1}, \ldots, g_{i_k}); g_i\rangle$ a *consistent pair*. In Akutsu et al. (1999a, 2000a, 2000b), the concept of consistency is defined a little differently. For each set of candidate parents, they check the consistency for each specific Boolean function. We call this *function consistency* (*f-consistency* for short) as distinguished from the *data consistency* (*d-consistency* for short) described above. If the data themselves are inconsistent as in Table 1, every Boolean function will fail to be consistent with them and we need not search for the Boolean functions for such cases any more. This point brings an important improvement in the efficiency i.e., it is much more efficient to search for the d-consistent parents first and then find the corresponding Boolean functions only for those parents. In this way, we can simply remove the super-exponential term $2^{2^k}$ which is over $4 \times 10^9$ when $k = 5$. This ‘data-first’ approach still can be applied to the trie structure algorithm (Atkutsu et al., 1999b) and we can also remove the super-exponential term there. The same improvement was obtained in Lähdesmäki et al. (2003), but we believe our presentation to be simpler and more intuitive.

The d-consistency is actually equivalent to the f-consistency in the sense that a $k$-tuple of parents is d-consistent with $g_i$ if and only if there exists a boolean function $f$ with $k$-inputs such that $f$ is f-consistent with $g_i$.

## 2.2. Time complexities

The simple exhaustive search algorithm checks the consistency for every possible tuple of candidate parents. We assume the number of consistent tuples is bounded by an unspecified

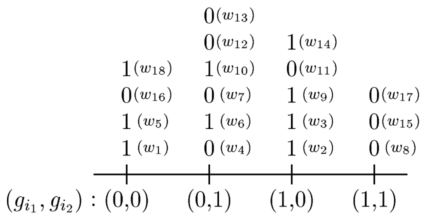

**Fig. 2** A tower shape diagram for noisy data. $w_j$'s are positive weights assigned to each measurement. To assign the best-fit mapping rule, we examine the towers one by one. The first tower, for example, has three 1's with total weight $w_{(0,0)}(1) = w_1 + w_5 + w_{18}$ and one 0 with weight $w_{(0,0)}(0) = w_{16}$, then for the input value $(g_{i_1}, g_{i_2}) = (0, 0)$, we assign 0 or 1 as the output value that has the larger total weight. When all the output values are assigned in this way, the best-fit Boolean function is determined

value $K \geq 1$ for each gene. $K$ may or may not depend on other variables depending on the network properties. For $k$-input parents, we examine $\binom{n}{k}$ candidate parents through the $m$ examples for each gene $g_i, i = 1, 2, \ldots, n$ and hence, the worst-case complexity is $O(mn^{k+1})$. However, in the average case, most candidate parents will quickly turn out to be inconsistent and we need not examine such profiles to the last $m$th data. We need to examine all the $m$ examples only for consistent parent tuple(s) ($\leq K$). The expected number of examples required to break the consistency for the uniformly distributed random input and output data is evaluated in Lemma 1 in Appendix. Hence, when we are searching for all the consistent parents for all genes, the average-case complexity is $O(n^{k+1} + Kmn)$.

## 2.3. Boolean networks for noisy data and the best fit extension problem

Real microarray data contain much noise—biological variation and the experimental measurement noise. Hence, we may not find even a single consistent tuple for a gene when the number of measurements is large. Hence, we allow some portion of inconsistent mappings. Here, we briefly formulate the so-called best-fit extension problem (Boros et al. 1998), (Lähdesmäki et al., 2003) in the data-first approach when the input number of parents are limited to $k$. We assign positive weights $w_j > 0, \quad i = 1, 2 \ldots, m$ for each measurement as in Lähdesmäki et al. (2003). Let $w = \sum_{j=1}^{m} w_j$ and $w_{(u_1,\ldots,u_k)}(0) :=$ sum of all the weights whose input value is $(u_1, \ldots, u_k) \in \{0, 1\}^k$ and the output value is 0. $w_{(u_1,\ldots,u_k)}(1)$ is defined similarly when the output value is 1. See Fig. 2 for an illustration. The *inconsistency ratio (= noise ratio)* IR is defined as

$$\mathrm{IR}(i_1, \ldots, i_k; i) := w^{-1} \sum_{(u_1,\ldots,u_k)\in\{0,1\}^k} \min[w_{(u_1,\ldots,u_k)}(0), w_{(u_1,\ldots,u_k)}(1)],$$

for the pair $\langle (g_{i_1}, \ldots, g_{i_k}); g_i \rangle$. We say the pair $\langle (g_{i_1}, \ldots, g_{i_k}); g_i \rangle$ is *consistent with noise-ratio nr* $\in (0, 0.5)$ if $\mathrm{IR}(i_1, \ldots, i_k; i) \leq nr$. For noisy data, we should examine the consistency of each candidate parents to the last $m$th example to estimate IR exactly. Hence, the time complexity is $O(mn^{k+1})$. We say a Boolean function is *the best-fit extension for the pair* $\langle (g_{i_1}, \ldots, g_{i_k}); g_i \rangle$ if it matches the observed mapping with the smallest IR. If an input state is missing in examples, the corresponding tower in Fig. 2 will be missing and the number of best-fit extensions will be doubled per missing of a tower.

Since assessing the weight of each measurement is actually impractical, we assume the uniform weight 1 for each measurement in this article.

## 3. Description of the top-down search algorithm

In this Section, we describe our main algorithm TDSA and derive an additional gain of $(\log m)^{(k-1)}$. We first evaluate the exact time complexity in the noise-free case. Keeping noisy data in mind, we consider the complexity of the average-case analysis for $n$, but of the worst-case analysis for $m$, because we should exploit all the $m$ data to exactly estimate the inconsistency ratios. TDSA is applicable for searching for multiple tuples of parents for each node, but we assume $K = 1$ in this Section to simplify our analysis.

### 3.1. Algorithm

(a) Superset selection : Observe that if a candidate tuple $G_i = (g_{i_1}, \ldots, g_{i_k})$ is consistent with $g_i$ then, any superset of $G_i$, $\tilde{G}_i = (g_{i_1}, \ldots, g_{i_k}, g'_{i_1}, \ldots, g'_{i_h})$, $h = 1, 2, \ldots$ is still consistent with $g_i$. We repeat the random selection until we have a consistent tuple of the cover size $cov = k + h$, and examine whether or not we can choose $k$ consistent subset within the consistent superset in the next step.

(b) Core search : For the consistent tuple $\tilde{G}_{ij} = (\tilde{g}_{ij}(1), \ldots, \tilde{g}_{ij}(cov))$, $i = 1, 2, \ldots, n$, $j = 1, 2, \ldots$, obtained in (a), we delete one gene in $\tilde{G}_{ij}$ in turn and check if each respective deleted tuple of size $cov - 1$ is consistent with $g_i$ or not. We make the *core vector* $C_{ij}$ of size $cov$ such that

$$C_{ij}(l) = \begin{cases} 1, & \text{if } \tilde{G}_{ij} \backslash \{\tilde{g}_{ij}(l)\} \text{ is not consistent with } g_i \\ 0, & \text{otherwise,} \end{cases}$$

for $l = 1, 2, \ldots, cov$. $C_{ij}(l) = 1$ means the $l$th element in $\tilde{G}_{ij}$ is indispensible to make the cover $\tilde{G}_{ij}$ consistent with $g_i$. We call $\tilde{g}_{ij}(l)$ a *core element* of $\tilde{G}_{ij}$ if $C_{ij}(l) = 1$ and let $n(C_{ij})$ be the number of the core elements in the selection $\tilde{G}_{ij}$. If $n(C_{ij}) > k$, $\tilde{G}_{ij}$ is not a superset of the $k$ parents and we need not examine such cases any more. Otherwise, we exhaustively choose $k - n(C_{ij})$ elements from the non-core elements and check if each choice added with the respective core elements constitute the consistent $k$ parents.

(c) Repeat (a) and (b) until we choose a consistent superset that contains the exact $k$ consistent subtuple. We take it as the $k$-parents for the gene $g_i$.

(d) We repeat (a), (b), and (c) for each $i = 1, 2, \ldots, n$.

### 3.2. Analysis

For one consistent parent tuple, the probability that a random selection of size $cov (> k)$ contains all the $k$ parents is

$$p = \binom{n-k}{cov-k} \Big/ \binom{n}{cov}$$

and the expected number of repetitions is

$$E_{rep} := \frac{1-p}{p} = \frac{n(n-1)\cdots(n-k+1)}{cov(cov-1)\cdots(cov-k+1)} - 1. \quad (1)$$

We let $N := 2^{cov}$ in this article. For a random superset of size *cov*, the probability that it firstly becomes inconsistent at $l$th example is evaluated in Lemma 1 in Appendix as

$$p_N(l) := \sum_{r=1}^{N} \binom{N-1}{r-1} 2^{r-1} S(l-1,r)\, r! \left(\frac{1}{2N}\right)^{l-1}$$

and hence, the probability of a candidate tuple of size *cov* to be consistent for $m$ examples is

$$p_{consis} := p_{consis}(cov, m) := \sum_{l=m+1}^{\infty} p_N(l). \tag{2}$$

Hence, on average, $p_{consis}$ portion of $E_{rep}$ will exhibit consistent tuples. For each such consistent *cov*-tuple, we take step (b) to examine if we can extract exact $k$ parents in the *cov*-tuple. The additional cost for constructing the core vectors is

$$p_{consis} \cdot E_{rep} \cdot cov(cov-1) \tag{3}$$

where *cov* is the number of search for each deleted tuple and $(cov-1)$ is the cost for the tuple size. If we directly search for $k$-subtuple instead of step (b), the cost (3) will be $p_{consis} \cdot E_{rep} \cdot \binom{cov}{k}$. It is $(cov-2)!/k!(cov-k)!$ $(k \geq 2)$ times more costly then (3). If the number of measurements are not sufficiently large, $p_{consis}$ will not be small and this difference becomes important. The cost for the sub-exhaustive search described in the latter part of (b) is

$$\begin{aligned} & p_{consis} \cdot E_{rep} \cdot k \sum_{i=0}^{k} \binom{cov}{i} q^i (1-q)^{cov-i} \binom{cov-i}{k-i} \\ &= p_{consis} \cdot k \sum_{i=0}^{k} \frac{1}{i!(k-i)!} q^i (1-q)^{cov-i} \cdot O(n^k). \end{aligned} \tag{4}$$

where $k$ is the tuple size cost and $q = q(cov, m)$ is the probability that $C_{ij}(l)$ is a core element i.e. a deleted tuple is not consistent with $g_i$ while the original tuple is consistent with $g_i$. $q$ is also evaluated in Lemma 3 in Appendix. Hence, summing up all the three costs (1), (3), and (4), the coefficient of $n(n-1)\cdots(n-k+1)$ for the overall computational cost is

$$\frac{cov(cov-k)!}{cov!} + p_{consis} \cdot \left\{ \frac{cov(cov-1)(cov-k)!}{cov!} + k \sum_{i=0}^{k} \frac{q^i(1-q)^{cov-i}}{i!(k-i)!} \right\}. \tag{5}$$

which depends on $cov,\ m$, and $k$. Let us choose the *cov*er size $cov = \lceil \log(m/2) \rceil$, then $p_{consis}$ in (2) still exponentially decreases to 0 and $1-q$ approximates to $2^{-N} \approx (1/\sqrt{2})^m$ when $m$ increases, which are shown in Lemma 2 and Lemma 3 respectively. Hence, the second and third terms become negligible and the above coefficient is $O(1/(\log m)^{(k-1)})$. Consequently, the overall average-case complexity for all the $n$ nodes is $O(n^{k+1}/(\log m)^{(k-1)})$ for the noise-free case.

### 3.3. Application of the algorithm in noisy setting

In noisy setting, we first determine the noise ratio $nr$ from data. For a *cov*er size $cov$, we regard candidate parents of size $cov > k$ are consistent with $g_i$ if IR$< nr_1$. Subsequently, we perform the core search and evaluate the core vector as in step (b). We regard $\tilde{g}_{ij}(l)$ is a core element if IR for $\tilde{G}_{ij} \backslash \{\tilde{g}_{ij}(l)\}$ is smaller than $nr_2$. And then, we perform a sub-exhaustive search for the exact $k$ parents and for the noise ratio $nr$. If the number of measurements $m$ is not sufficiently large, we recommend taking the noise ratios such that $nr \le nr_2 \le nr_1$ by small increments, because the estimation of noise ratios can be less accurate for candidate parents with large *cov*er size.

To show that the proposed algorithm has the same gain in complexity for noisy data, we need to show that $p_{consis}$ in (5) still exponentially decreases to 0 if we choose $cov = O(\log m)$. Here, we turn to the f-consistency temporarilly. For a specific Boolean function with $k$ inputs, the probability that it is consistent with $nr$ is $P(B \le nr \cdot m)$ for a binomial random variable $B = B(m, 1/2)$ where 1/2 is the probability that an example is f-consistent. Hence, by Markov inequality we have

$$\begin{aligned} &P(\text{a superset tuple is d-consistent with } g_i) \\ &\qquad = P(\text{one of } 2^{2^{cov}} \text{Boolean function is consistent with } g_i) \\ &\qquad \le 2^{2^{cov}} \cdot P(\text{a Boolean function is consistent with } g_1) \\ &\qquad = 2^{2^{cov}} \cdot P(B \le nr \cdot m) \\ &\qquad = 2^{2^{cov}} \cdot P(B \ge (1 - nr)m) \\ &\qquad \le 2^{2^{cov}} \cdot \left[ (nr \cdot e + 1 - nr) e^{-(1-nr)} \right]^m . \end{aligned}$$

Since $(nr \cdot e + 1 - nr)e^{-(1-nr)} < 1$ for $0 < nr < 0.4$, the above probability still exponentially decreases to 0 if we choose $2^{cov} \approx \sqrt[a]{m},\ a > 1$, i.e. $cov = O(\log m)$. In this noisy case, the average time complexity is $O(mn^{k+1}/(\log m)^{(k-1)})$.

## 4. Computational experiments

In this Section, we tested our main algorithm for the new gain of $(\log m)^{(k-1)}$. We named the improved algorithm given in Section 2 *d-naive algorithm*. We compared the CPUTIMEs taken by d-naive algorithm and TDSA given in Section 3 using their ratios of CPUTIMEs for several $cov$, $m$, and $k$. Because d-naive algorithm has the same time complexity as the algorithm of Lähdesmäki et al. (2003), tests in this Section will address the comparison with the best previous method (Lähdesmäki et al., 2003). Other efficient algorithms (Akutsu et al, 2003; Fukagawa and Akutsu, 2003) are applicable only for some subclass of Boolean functions, while TDSA is applicable to the whole function class. Moreover, while the greedy algorithm (Akutsu et al., 2003) was incomparably fast with linear time complexity for $n$, the accuracy was far from satisfactory. In the tests for $m = 100$, and $k = 3$ and 4, the greedy algorithm showed very poor accuracy of 83% and 2% for the noise free case (Akutsu et al., 2003). However, TDSA showed 100% (80/80) and 98.75% (79/80) accuracy even with 5% IR. For other measurements ($m \ge 200$), TDSA found the underlying Boolean relationships with 100% accuracy.

Because learning Boolean networks can be done independently gene by gene, we consider learning Boolean relationships for individual genes in our test. We fix $n = 50$ and test the following seven logical functions for $k = 1, 2, 3$, and 4 that determine the expression levels of $g_1 \sim g_7$ respectively.

$$
\begin{aligned}
g_1 &= g_{15} \wedge g_{20} \\
g_2 &= g_{18} \vee g_{22} \\
g_3 &= (g_{25} \wedge g_{33}) \bar{\vee} g_{40} \\
g_4 &= \neg(g_{30} \wedge g_{41}) \wedge g_{45} \\
g_5 &= (g_{17} \wedge g_{24}) \bar{\vee} (g_{24} \wedge g_{38}) \bar{\vee} (g_{38} \wedge g_{42}) \bar{\vee} (g_{17} \wedge g_{42}) \\
g_6 &= \neg(g_{14} \wedge g_{27}) \wedge (g_{29} \vee g_{46}) \\
g_7 &= g_{12}.
\end{aligned}
$$

We generated other expression levels of $g_8 \sim g_{50}$ uniformly at random. Because TDSA is a randomized algorithm, we repeated each computation 40 times and took the average CPUTIMEs. For the CPUTIME of d-naive algorithm, we halved the worst-case CPUTIME taken for the exhaustive $\binom{n}{k}$ search to obtain the average cost for $n$. For noise, we altered 5% of the output signals. Computation results for $g_4$ ($k = 3$ and $m = 200$) are shown in Fig. 3. The minimal cost was obtained when $cov = 10$ and $nr_1 = 0.1$. If a smaller cover size is taken, the probability of the superset to include the true parents decreases, and the expected number of superset selection is increased. Conversely, if the larger cover size is taken, the probability increases, but more false positive supersets that do not contain the $k$ parents should be examined. This caused the U-shape costs in Fig. 3. The choice of $nr_1$ was also important for the performance of TDSA. Figure 3 suggests TDSA gives robust and good performance for the wide range of the choices of cover size and $nr$.

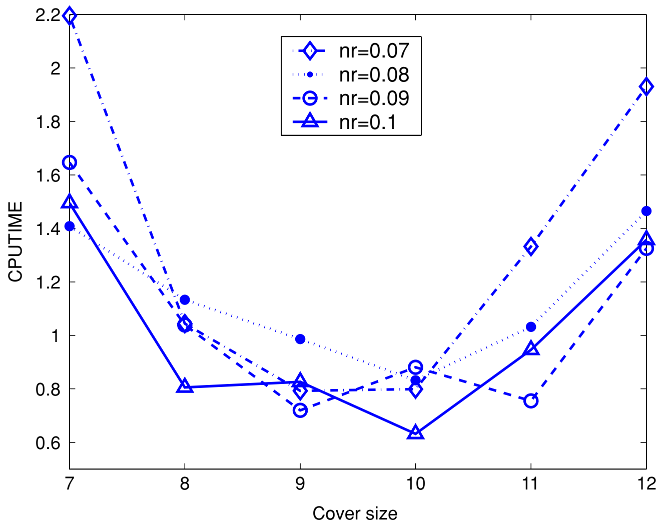

**Fig. 3** Cost plots of the case $k = 3$ and $m = 200$ for various *cov*er sizes and $nr_1$'s

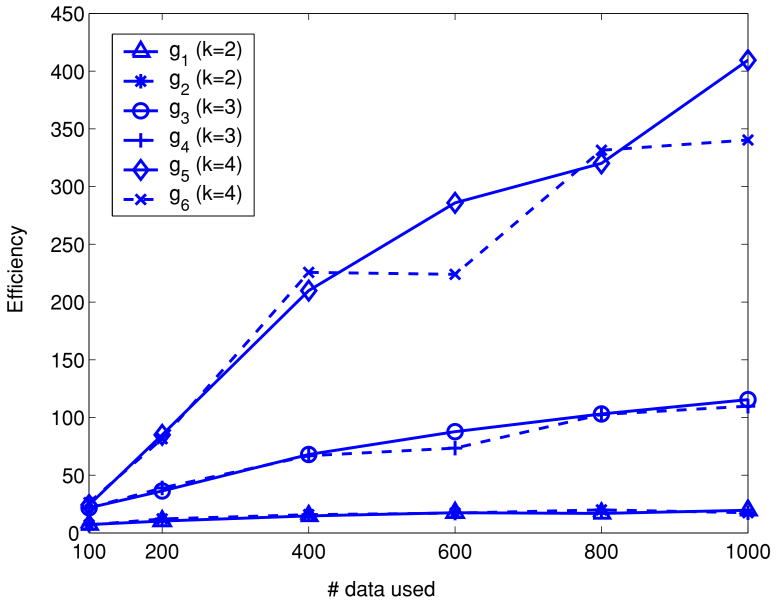

**Fig. 4** Efficiency comparison for Boolean functions

To compare the performances, We define

$$\textbf{Efficiency} := \frac{0.5 \cdot \text{worst-case CPUTIME of d-naive algorithm}}{\text{CPUTIME of TDSA}}. \tag{6}$$

In Fig. 4, the Efficiencies are depicted for $g_1 \sim g_6$. They get larger if $m$ and $k$ are increased and TDSA was from dozens to hundreds times faster than d-naive algorithm. Note that since the Efficiency is $O((\log m)^{(k-1)})$, it exponentially grows with $k$ while it grows by log scale for $m$. The test result for $g_7$ ($k = 1$) is shown in Fig. 5. In this case, TDSA was on average $1.6 \sim 1.7$ times faster, but the Efficiency was not increased with the increase of $m$ as we expected. For $k = 1$, the gain $O((\log m)^{(k-1)})$ is $O(1)$ and this test confirms again that our new gain is exact. Because the cover size is taken for $O(\log m)$, the optimal cover size slowly increased with the increase of $m$ except for the case of $k = 1$ (Fig. 6).

The comparison of the cases $n = 50$ and $n = 100$ is shown for $g_2$ and $g_4$ in Fig. 7. The optimal cover sizes for $g_2$ and $g_4$ ranged from 7 to 12 and 8 to 13, respectively both for $n = 50$ and $n = 100$ cases implying that the optimal cover size is not affected by $n$. Therefore, we can quickly estimate the optimal cover sizes for small number of sampled nodes before we learn the Boolean networks of all nodes.

We also tested the case of IR $= 0.1$ for $g_2$ and $g_4$. For higher noise ratio, more false-positive supersets can be consistent for target genes without true parent nodes. This made the Efficiencies a little reduced compared to the cases of IR $= 0.05$ (Fig. 8).

Lastly, we measured the success rates of TDSA. When we chose the optimal cover size and $nr_1$'s, TDSA precisely found the parent nodes of the seven target nodes in all 40 independent trials for all the numbers of measurements tested, except for the one case of $g_6$ ($k = 4$) and $m = 100$ where 39 trials correctly found the four parent nodes. Since TDSA checks the consistency of $k$ candidate parents at the final step, it gives no false-positive predictions. The

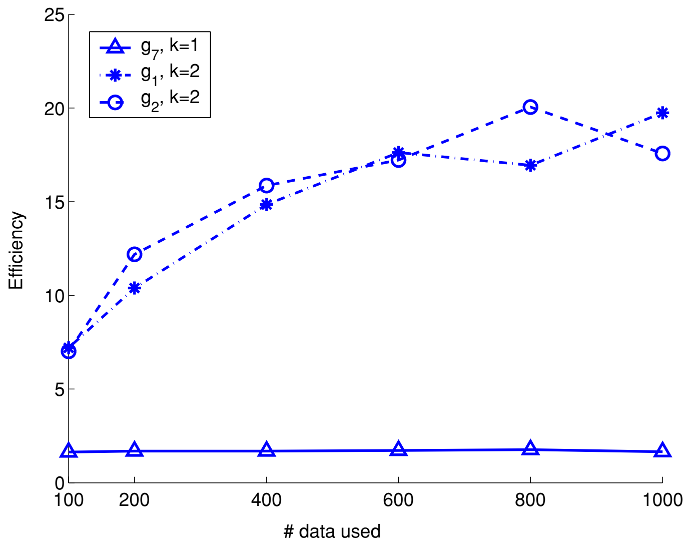

**Fig. 5** Efficiency comparison for one regulator case

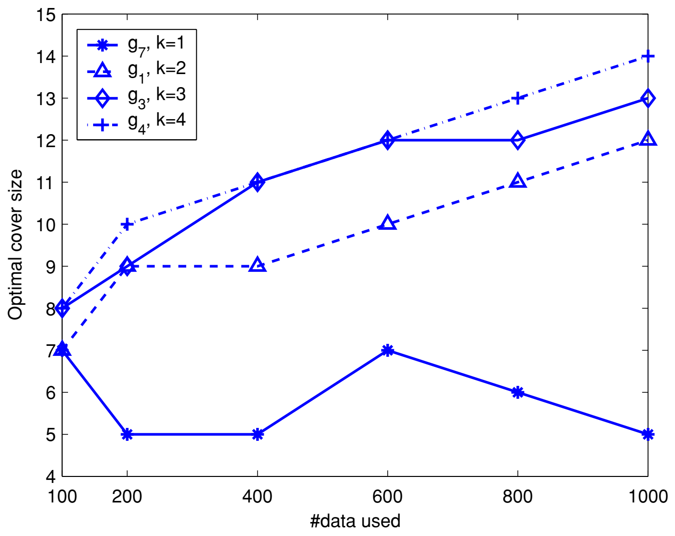

**Fig. 6** Optimal cover sizes. For each measurement, the cover sizes that yielded the optimal costs (e.g. $cov = 10$, $nr_1 = 0.1$ in Fig. 3) were shown. The optimal cover size clearly increased with $m$ for $k = 2$, 3 and 4 while no pattern was observed for $k = 1$

success rate is generally increased if the cover size is decreased and vice versa. In Fig. 9, the success rates averaged on different $nr_1$'s are shown for $g_1$ ($k = 2$).

The test results suggest that the parameters (cover size and $nr_1$) chosen to maximize efficiency also provide accurate results. With a large number of measurements a large cover size can be used to obtain fast and accurate results, while smaller cover size should be used with a small number of measurements to obtain accurate results at the cost of efficiency.

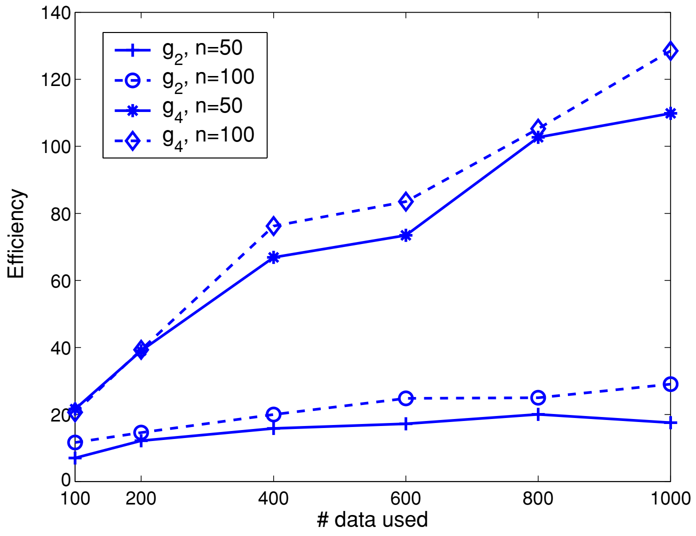

**Fig. 7** Efficiency comparison for different number of nodes

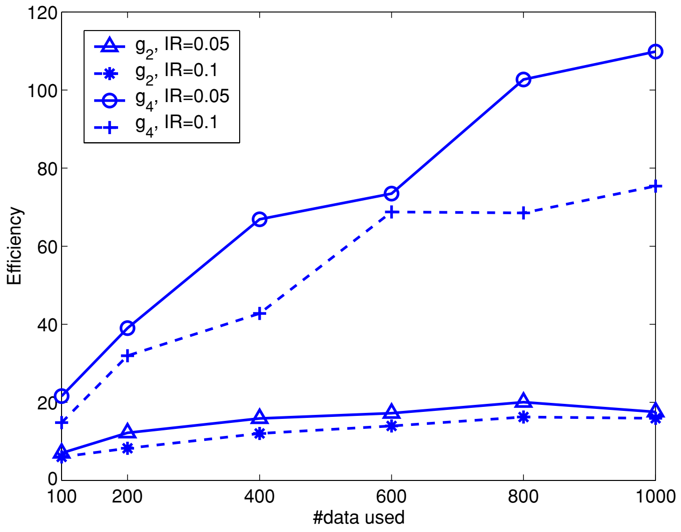

**Fig. 8** Efficiency comparison for Inconsistency Ratio

## 5. Learning Boolean networks from real expression data

In the previous section, we tested TDSA for relatively small IRs and $K = 1$ to investigate the various aspects of the new algorithm. However, real microarray data contain much higher rate of noise, and a number of multiple tuples of parents are detected for given IRs. Additionally, the quantized expression data are not uniformly distributed. Using yeast expression data, we tested TDSA for this composite situation. For the case of multiple tuples, we repeated TDSA until all the consistent parent tuples for given IR were caught.

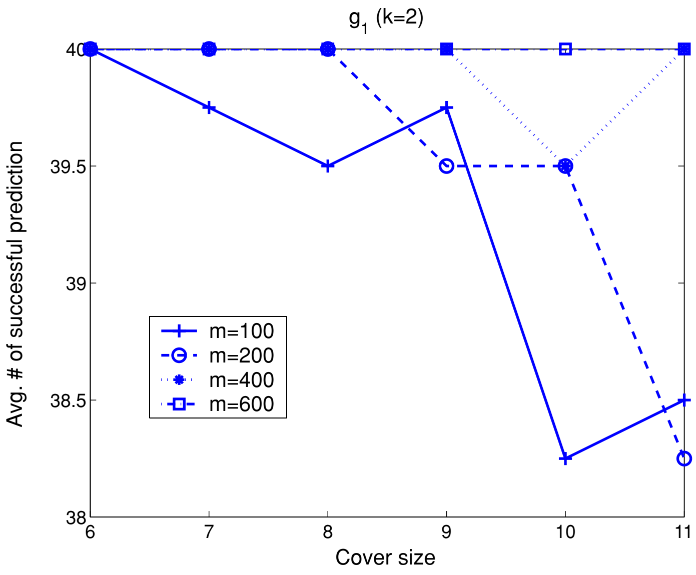

**Fig. 9** #successful predictions for different cover sizes

We combined three microarray data sets from Spellman et al. (1998), (Gasch et al., 2000), and (Gasch et al., 2001) for total 302 measurements, and quantized the data at 0. We chose seven genes, Clb2, Cdc20, Cln1, Cln2, Bud4, Bud9, and Rax2 (Table 2) involved in cell cycle control or budding, and examined how these genes are predicted by the expression levels of the 111 transcription factors (TF's) chosen from Lee et al. (2002) that have expression profiles in the three data sets). For most cases, higher IR's of ∼20% and multiple tuples of parents were detected.

We tested how fast TDSA finds all the true parent tuples without loss of accuracy. In this highly noisy data, the optimal cover sizes were smaller (cover size = 5 and 6, for $k = 2$ and $k = 3$ respectively) than the small noise cases (5∼10% IR) and noise ratio $nr_1$ = IR gave the optimal performance. These parameter choices gave 100% accuracy. The test results are summarized in Table 2. We compared the CUPTIME's taken for exhaustive search by d-naive algorithm and TDSA. We generalize the definition of Efficiency (6) for multiple tuples of parents as follows:

$$\textbf{Efficiency}(\mathbf{r}) := \frac{\frac{r}{r+1}\cdot \text{worst-case CPUTIME of d-naive algorithm}}{\text{CPUTIME of TDSA for all the } r \text{ tuples}}$$

for $r$ tuples of parents. We evaluated Efficiencies for less than 100 parent tuples for each gene. The Efficiencies were 3.6 ∼ 4.8 for $k = 2$, and 6.8 ∼ 12.6 for $k = 3$. Note that the Efficiencies for 5% IR and $m =\sim 300$ were 13 ∼ 14 for $k = 2$, and ∼50 for $k = 3$. The Efficiency for Clb2 for parent tuple number $\leq$ 1000 is depicted in Fig. 10, where the Efficiency decreased with the increase of the tuple number. Hence, before applying TDSA, we need to control IR such that we may not have too many parent tuples. In short, TDSA is more efficient for smaller rate of noise and smaller number of parent tuples.

**Table 2** Test results for seven genes. The numbers of parent tuples that are consistent for given IR's are shown. In parentheses, corresponding CPUTIMEs taken for finding all the parent tuple(s) by TDSA are shown. The parentheses in Indegree column denote the CUPTIMEs for d-naive algorithm

| Indegree | Gene\ IR | 0.18 | 0.19 | 0.20 | 0.21 | 0.22 | 0.23 |
|---|---|---|---|---|---|---|---|
| $k = 2$ (23.34) | Clb2 | 0 | 0 | 0 | 0 | 38 (6.33) | 109 (11.05) |
| | Cdc20 | 0 | 0 | 0 | 3 (3.61) | 18 (4.18) | 105 (8.45) |
| | Cln1 | 0 | 0 | 0 | 0 | 0 | 0 |
| | Cln2 | 0 | 0 | 0 | 0 | 0 | 0 |
| | Bud4 | 0 | 0 | 0 | 0 | 0 | 0 |
| | Bud9 | 0 | 0 | 0 | 0 | 21 (4.73) | 110 (8.65) |
| | Rax2 | 0 | 0 | 0 | 0 | 2 (4.93) | 86 (9.67) |
| $k = 3$ (863.55) | Clb2 | 7 (69.99) | 38 (79.44) | 228 | 748 | 2966 | 6208 |
| | Cdc20 | 0 | 1 (63.55) | 88 (79.26) | 423 | 1986 | 5658 |
| | Cln1 | 0 | 0 | 0 | 0 | 1 (44.32) | 5 (73.29) |
| | Cln2 | 0 | 0 | 2 (70.34) | 5 (62.25) | 20 (92.97) | 78 (135.60) |
| | Bud4 | 0 | 23 (69.26) | 564 | 5092 | 5996 | 6002 |
| | Bud9 | 0 | 0 | 8 (61.12) | 142 | 2305 | 5834 |
| | Rax2 | 1 (59.60) | 7 (69.75) | 42 (90.78) | 140 | 738 | 4576 |

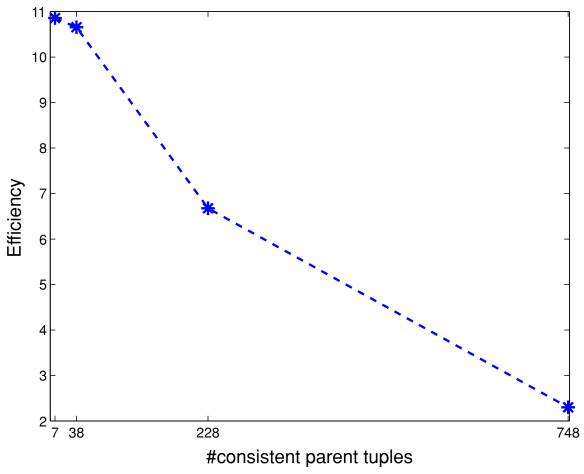

**Fig. 10** Efficiencies for learning multiple parent tuples of Clb2 ($k = 3$)

## 6. Conclusion

We proposed two independent ways to improve network search efficiencies, and our total gain was $2^{2^k}(\log m)^{(k-1)}$. For optimal implementation of our algorithm, the cover size *cov* and the noise ratio *nr* was chosen appropriately for each number of measurements and input parents. These can be estimated from tests for sampled nodes.

Boolean networks of gene expression represent consistent logical relationships among expression profiles. Therefore, they may not necessarily reflect known biological interactions such as transcriptions or protein interactions that occur in different stages of biological processes. Such gene expression networks may be interpreted as remote, but tightly organized, biological relationships that are determined by complex activities of many biological objects.

Among many biological interactions, transcription networks may most resemble the gene expression networks. In our test, however, only one of the seven genes (Bud9) had high scoring Boolean relationships enriched in location analysis data. Although previous pilot studies have been done by Lähdesmäki et al. (2003), there are still difficulties in mapping gene expression networks to gene regulatory networks. As shown in section 5, microarray expression data contain higher rate of noise, and a number of multiple tuples of consistent parents are detected for given IR's. Two other difficulties stem from the limitation of the mathematical model: (1) data quantization causes information loss, and (2) only a limited number of parents should be considered due to computational and experimental reasons. The latter provides another source of noise: the activity of hidden regulators. Despite these challenges, the innate simplicity and intuitiveness of the model make Boolean networks strong candidate methods for learning regulatory networks, as heterogeneous sources of information for proteins and metabolites become available.

## Appendix: Some discrete probability calculations

Let us consider the Fig. 1. Suppose the slots are $N$ people and 0 and 1 represent two different coupons. Both probabilities for each coupon are equally 1/2 and the coupons are distributed uniformly randomly to $N$ people one by one. The process stops when a person firstly collect both the coupons 0 and 1. Then, the probability that the process stops at $m$th coupon and its expectation are given in the following Lemma.

*Lemma 1.* (Two-coupon collection problem for $N$ people). Let $M$ be the random variable of the number of coupons distributed when the process explained above stops, then the probability distribution of M is given as

$$P_N(M = m) := \sum_{r=1}^{N} \binom{N-1}{r-1} 2^{r-1} S(m-1, r)\, r! \left(\frac{1}{2N}\right)^{m-1}, \quad m \geq 2,$$

where $S(n, k)$ is the Stirling number of the second kind, and its expectation is

$$E_N = \sum_{r=1}^{N} \binom{N-1}{r-1} 2^{r-1} r! \frac{(2N-r-1)!}{(2N-1)!} \sum_{l=0}^{r} \frac{2N}{2N-l}.$$

**Proof:** We give a combinatorial proof here. The meanings of each term are

- We suppose the coupon distribution stopped at $m$th example with coupon '1' at the $N$th slot.
- $r$ represents the number of slots occupied when the process stoped.
- Except for the slot where the process ended, we should choose $r-1$ slots occupied among the $N-1$ slots : $\binom{N-1}{r-1}$.
- The $r-1$ slots chosen have two choices 0 or 1 : $2^{r-1}$.
- We partition the former $m-1$ coupons into $r$ groups and distribute each group to the $r$ slots : $S(m-1,r)\,r!$.
- All possible choices except for the last coupon : $(1/2N)^{m-1}$.

Hence, using the generating function for $S(\cdot,k)$,

$$\sum_n S(n,k)\,x^n = \frac{x^k}{(1-x)(1-2x)\cdots(1-kx)}, \quad k \geq 0$$

(see Wilf, 1992), the expected number of coupons required is evaluated as

$$\begin{aligned} E_N &= \sum_{m\geq 2} m \cdot P_N(M=m) \\ &= \sum_{r=1}^{N} \binom{N-1}{r-1} 2^{r-1} r! \sum_{m\geq 2} m \cdot S(m-1,r) \left(\frac{1}{2N}\right)^{m-1} \\ &= \sum_{r=1}^{N} \binom{N-1}{r-1} 2^{r-1} r! \frac{(2N-r-1)!}{(2N-1)!} \sum_{l=0}^{r} \frac{2N}{2N-l}. \end{aligned}$$

□

*Lemma 2.* $p_{consis}$ in (2) exponentially decreases to 0 when $m \uparrow \infty$. $p_{consis}$ still exponentially decreases to 0 if we choose $N = \lceil m/2 \rceil$ i.e. $cov = O(\log m)$.

**Proof:** The Stirling number $S(n,k)$ in (2) is expressed as

$$S(n,k)\,k! = \sum_{i=0}^{k} \binom{k}{i} (-1)^i (k-i)^n,$$

which represents the number of onto functions from $n$ elements to $k$ elements (see Stanley, 1997). Hence, asymptotically we have

$$\lim_{n\to\infty} S(n,k)/k^n = 1/k!, \tag{7}$$

and using this fact, we have

$$p_{consis} \approx \sum_{r=1}^{N} \binom{N-1}{r-1} 2^{r-1} \frac{2N}{2N-r} \left(\frac{r}{2N}\right)^m,$$

and its upper bound as

$$\begin{aligned}&\leq \left(\frac{1}{2}\right)^m \sum_{r=1}^{N} \binom{N-1}{r-1} 2^{r-1}\\ &= (1/2)^m \cdot 3^{N-1}.\end{aligned}$$

Hence, $p_{consis}$ exponentially decreases to 0 if we choose $N = \lceil m/2 \rceil$ when $m \uparrow \infty$. □

*Lemma 3.* For $q$ in (5), $1 - q$ approximates to $2^{-N}$ when $m \uparrow \infty$.

**Proof:** $q$ in (4) and (5) is the probability that a deleted $(cov - 1)$-tuple is inconsistent with a gene $g_i$ conditioned that the original $cov$-tuple is consistent with $g_i$. Let us consider the following problem first. Under the condition that we chose $i$ slots among $2N \geq i$ slots, the probability that we have chosen $d$ pairs of slots of the form $(j, N + j)$, $1 \leq j \leq N$ is given as

$$p_{N,i}(d) = \binom{2N}{i}^{-1} \binom{N}{d} \binom{N-d}{i-2d} 2^{i-2d}, \quad 1 \leq d \leq \lfloor i/2 \rfloor,$$

where the respective terms mean in sequence the inverse of the total number of possible cases, possible number of pair choices, the number of choices among $i$ selections which does not form pairs, and the possible choices of the positions for the third term between the two $j$th and $(N + j)$th slots.

The second problem to consider is as follows: Suppose the $m$ examples were consistent for $2N$ slots. Then, the probability that the number of occupied slots is $i$ is given by Bayes rule as

$$\begin{aligned}p_N(i|m) &= P(i \text{ slots are occupied} \mid m \text{ examples are consistent on } 2N \text{ slots})\\ &= \binom{2N}{i} S(m,i)\, i!\, 2^i \Big/ \sum_{j=1}^{2N} \binom{2N}{j} S(m,j)\, j!\, 2^j, \quad 1 \leq i \leq 2N.\end{aligned}$$

Hence, the probability for the number of pairs when the $m$ examples are consistent is

$$p_N(d) = \sum_{i=2d}^{N+d} p_{N,i}(d) \cdot p_N(i|m)$$

and the deleted tuple is consistent with probability

$$p_N = \sum_{d=1}^{N} 2^{-d} \cdot p_N(d).$$

Hence,

$$q = 1 - p_N.$$

Using the fact (7), we can show $p_N(2N|m) \rightarrow 1$ as $m \uparrow \infty$, and hence $p_N(N) \rightarrow 1$ while $p_N(d) \rightarrow 0$ for all $d < N$, and we have

$$1 - q = p_N \rightarrow 2^{-N}.$$

□

**Acknowledgements** This work was supported by KRIBB research initiative program. The authors wish to thank Dr. Akutsu for sharing the references (Akutsu et al., 1999b) and (Fukagawa and Akutsu, 2003) as preprints as well as for helpful comments, and thank Dr. Lähdesmäki for helpful comments for Lähdesmäki et al. (2003).

## References

Akutsu, T., Kuhara, S., & Miyano, S. (1999a). Identification of genetic networks from a small number of gene expression patterns under the Boolean network model. *Proc. Pacific Symp. on Biocomputing '99*, World Scientific, (pp. 17–28).

Akutsu, T., Miyano, S., & Kuhara, S. (1999b). Fast identification of Boolean networks. *Technical Report 99-AL-66*, Information Processing Society of Japan: (pp. 25–32).

Akutsu, T., Miyano, S., & Kuhara, S. (2000a). Algorithms for identifying Boolean networks and related biological networks based on matrix multiplication and fingerprint function. *J. Comp. Biol.*, *7*(3/4), 331–343.

Akutsu, T., Miyano, S., & Kuhara, S. (2000b). Inferring qualitative relations in genetic networks and metabolic pathways. *Bioinformatics*, *16*(8), 727–734.

Akutsu, T., Miyano, S., & Kuhara, S. (2003). A simple greedy algorithm for finding functional relations: efficient implementation and average case analysis. *Theoretical Computer Science*, *292*(2), 481–495.

Fukagawa, D., & Akutsu, T. (2003). Performance analysis of a greedy algorithm for inferring Boolean functions *Discovery Science*.

Friedman, N., Linial, M., Nachman, I., & Pe'er, D. (2000). Using Bayesian networks to analyze expression data. *J. Comp. Biol.*, *9*, 601–620.

Kauffman, S.A. (1969). Metabolic stability and epigenesis in randomly constructed genetic nets. *J. Theor. Biol.*, *22*, 437–466.

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., & Somogy, R. (1998). REVEAL. a general reverse engineering algorithm for inference of genetic network architectures. *Proc. Pacific Symp. on Biocomputing '98*, World Scientific, (pp. 18–29).

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

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

Lähdesmäki, H., Shmulevich, I., & Yli-Haria, O. (2003). On learning gene regulatory networks under the Boolean network model. *Machine Learning*, *52*, 147–167.

Stanley, R. P. (1997). Enumerative Combinatorics (vol. 1). Cambridge Press.

Wilf, H. S. (1992). Generatingfunctionology. Academic Press.