アルゴリズム 70－4
（1999. 11. 8）

行列乗算とハッシュ関数を用いたブーリアンネットワークの同定アルゴリズム

阿久津 達也* 宮野 悟* 久原 哲#

*東京大学医科学研究所ヒトゲノム解析センター
#九州大学大学院生物資源環境科学研究科

遺伝子ネットワークの推定問題のモデル化の一つとしてブーリアンネットワークの同定問題が研究されてきたが、この問題は複数のブール関数を同時に同定する問題としても定式化することができる。変数(頂点)の個数を n、次数制約を D、例の個数を m とする時に、これまで、$O(mn^{D+1})$ 時間アルゴリズムおよび $O(mn^D)$ 時間アルゴリズムが開発されてきた。本研究では、文字列マッチングに用いられる確率的なハッシュ関数を用いて行列乗算問題に帰着させることにより、この問題に対する $O(m^{\omega-2}n^D + mn^{D+\omega-3})$ 時間の確率的アルゴリズムを開発した。なお、ω は $n \times n$ の行列乗算の時間計算量における n のべきであり、現在のところ $\omega < 2.376$ である。

Algorithms for Identifying Boolean Networks Based on Matrix Multiplication and Fingerprint Function

Tatsuya Akutsu* Satoru Miyano* Satoru Kuhara#

*Human Genome Center, Institute of Medical Science, University of Tokyo
4-6-1 Shirkanedai, Minato-ku, Tokyo 108-8639, Japan
{takutsu,miyano}@ims.u-tokyo.ac.jp

Graduate School of Genetic Resources Technology, Kyushu University
6-10-1 Hakozaki, Higashi-ku, Fukuoka 812-8581, Japan
kuhara@grt.kyushu-u.ac.jp

This paper studies efficient algorithms for identifying Boolean networks of bounded indegree and related problems, where identification of a Boolean network is formalized as a problem of identifying many Boolean functions simultaneously. For the identification of a Boolean network, an $O(mn^{D+1})$ time naive algorithm and a simple $O(mn^D)$ time algorithm are known, where n denotes the number of nodes, m denotes the number of examples, and D denotes the maximum indegree. This paper presents an improved $O(m^{\omega-2}n^D + mn^{D+\omega-3})$ time Monte-Carlo type randomized algorithm, where ω is the exponent of matrix multiplication (currently, $\omega < 2.376$). The algorithm is obtained by combining fast matrix multiplication with the randomized fingerprint function for string matching.

1 Introduction

Deriving Boolean relations or other functional relations is important in computational molecular biology [3, 7, 13] and relational database systems [16, 17]. In molecular biology, inference of a genetic network architecture from time series of gene expression patterns becomes very important, due to recent progress of the DNA microarray technology. In relational database systems, inference of functional dependencies from data is useful for analyzing data and reducing disk space [16, 17].

Some studies have been done on the inference of genetic networks from state transition data using the *Boolean network* model [3, 13]. Liang, Fuhrman and Somogyi [13] proposed a heuristic algorithm for inference of Boolean networks of bounded indegree. Although they did not analyze the time complexity, it seems that the worst case complexity is $O(mn^{D+1})$, where m is the number of examples, n is the number of nodes, and D is the maximum indegree. We also proposed a simple $O(mn^{D+1})$ time algorithm in order to analyze the sample complexity [3]. Then, we developed an improved $O(mn^D)$ time algorithm [4]. On the other hand, we can prove that identification of a Boolean network is NP-hard if D is not a constant (i.e., D is included in an input). Therefore, it seems very difficult to design an algorithm for which the exponent is much smaller than D. In this paper, we present an improved $O(m^{\omega-2}n^D + mn^{D+\omega-3})$ time Monte-Carlo type randomized algorithm, where ω is the exponent of matrix multiplication (currently, $\omega < 2.376$ [6]). Although it is a slight improvement and the algorithm and its analysis are simple, the result is non-trivial. The algorithm is obtained by reducing the identification problem to matrix multiplication, using the randomized fingerprint function [11]. The technique is also applied to the identification of *functional dependencies* in a fixed domain and the identification of *qualitative relations*, where application to the latter one is omitted in this paper. Therefore, this paper shows new applications of matrix multiplication and the randomized fingerprint function.

Since a Boolean network can be considered as a set of Boolean functions, previous algorithms developed for inferring a Boolean function [12, 15] might be applied to the identification of Boolean networks. In particular, the WINNOW algorithm [14] is simple and practical for inferring Boolean functions with a few variables. However, in order to apply the WINNOW algorithm to the identification of Boolean networks of bounded indegree, some post-processing would be required. It seems that post-processing will take $O(mn^{D+1})$ time in the worst case, using a simple algorithm (Note that we are interested in the worst case time complexity in this paper). Of course, the algorithms proposed in this paper can also be used for post-processing.

Some algorithms were developed for inferring functional dependencies [16, 17]. Although the developed algorithms are general ones, the worst case time complexities seem to be $O(mn^{D+1})$ if they are modified for inferring functional dependencies with at most D input attributes.

2 Problems and Results

In this paper, we consider three types of problems:

CONSISTENCY: Decide whether or not there exists a Boolean network (resp. function) consistent with the given examples, and output one if it exists,

COUNTING: Count the number of Boolean networks (resp. functions) consistent with the given examples,

IDENTIFICATION: Decide whether or not there exists a *unique* Boolean network (resp. function) consistent with the given examples, and output it if it exists.

Although we present algorithms for the *counting problem*, they can be converted for the consistency problem and the identification problem without increasing the order of the time complexity. We formally define the *counting problem* for Boolean networks (resp. functions) as follows.

INPUT: $y_j(k)$ $(j = 1 \ldots l,\ k = 1 \ldots m)$, $x_i(k)$ $(i = 1 \ldots n,\ k = 1 \ldots m)$, and integer D, where $x_i(k)$ and $y_j(k)$ take Boolean values (i.e., 0 or 1) respectively,

OUTPUT: for each j, the number of Boolean functions $f_j(x_{i_1}, \ldots, x_{i_D})$'s such that $y_j(k) = f_j(x_{i_1}(k), \ldots, x_{i_D}(k))$ holds for all $k = 1 \ldots m$.

We call a tuple $\langle y_1(k), \ldots, y_l(k), x_1(k), \ldots, x_n(k) \rangle$ for each k as an *example*.

This problem is NP-hard for general D (i.e., D is included in INPUT), which can be proved as in [2, 15, 17]. Therefore, we are interested in the case where D is a constant. Particularly, we are interested in the case of $D = 2$ since we can reduce higher-dimensional problems to two-dimensional problems by using a simple method described in Section 3.2. Moreover, we are interested in the case of $l = 1$ and the case of $l = n$. For the case of $l = 1$, it is the identification problem of a Boolean function. For the case of $l = n$, it is the identification problem of a Boolean network. For a general case of the problem, there is a trivial algorithm: for each y_j, for all type of 2^{2^D} Boolean functions f, for all combinations of D variables $x_{i_1}, \ldots, x_{i_D}$, examine whether or not $y_j(k) = f(x_{i_1}(k), \ldots, x_{i_D}(k))$ holds for $k = 1 \ldots m$. This algorithm takes $O(lmn^D)$ time for a constant D. *Note that, although we should carefully count the number of functions so that the same function is not counted more than once, it can be done without affecting the orders of the time complexities in all algorithms presented in this paper.*

Recently, we developed an improved $O(mn^D + lm)$ time algorithm [3], by using a *trie*, which is a well-known data structure in string matching [1]. In this paper, we further improve the time complexity and we show the following algorithms:

- An $O(n^D m^{\omega-2} + n^{D+\omega-3}m)$ time deterministic algorithm for the case of $l = 1$,
- An $O(n^D m^{\omega-2} + n^{D+\omega-3}m)$ time Monte-Carlo type randomized algorithm for the case of $l = n$,

where ω denotes the exponent of matrix multiplication (i.e., matrix multiplication of $n \times n$ matrices can be done in $O(n^\omega)$ time) [6]. Note that, for $m \times n$ matrices X and Y, matrix product $X \cdot Y^t$ can be computed in $O(mn^{\omega-1})$ time if $m \geq n$, otherwise it can be computed in $O(m^{\omega-2}n^2)$ time, by partitioning each matrix into small square matrices. Note also that, in this paper, Y^t denotes the transposed matrix of Y. Recently, some improvement on matrix multiplication was done for the case of $m \neq n$ [9]. That result might be useful for improving the time complexities of the algorithms in this paper for the case of $m \ll n$.

Algorithms robust for noises and an approximation algorithm for general D are shown too. Although Boolean functions are considered in the above, the algorithm for the case of $l = n$ can be extended for finding functional relations (or, functional dependencies) in a fixed domain.

3 Algorithms

In this section, we describe algorithms for the counting problem for Boolean functions and Boolean networks. Before describing the algorithms, we note that Boolean functions $f(x, y)$ with two input variables are classified into the following categories:

CONSTANT: 0, 1, **UNARY:** x, $\overline{x}$, y, $\overline{y}$, **XOR:** $x \oplus y$, $\overline{x \oplus y}$,
AND: $x \wedge y$, $x \wedge \overline{y}$, $\overline{x} \wedge y$, $\overline{x} \wedge \overline{y}$, **OR:** $x \vee y$, $x \vee \overline{y}$, $\overline{x} \vee y$, $\overline{x} \vee \overline{y}$.

In this paper, different types of Boolean functions are counted separately. Since the countings of CONSTANT and UNARY functions are easier, we only consider AND, OR, XOR functions.

3.1 An Algorithm for Boolean Functions

In this subsection, we consider the case of $l = 1$. First we show an algorithm for counting the number of Boolean functions of the form $x \wedge \overline{y}$. The other types of functions in AND and OR categories can be counted in a similar way.

Counting of $x \wedge \overline{y}$ functions

The algorithm consists of two steps. The first step is similar to the PAC learning algorithm for monotone Boolean functions [12, 19]. It begins with the conjunction of all literals

$$x_1 \wedge x_2 \wedge \ldots \wedge x_n \wedge \overline{x_1} \wedge \overline{x_2} \wedge \ldots \wedge \overline{x_n},$$

and processes examples one by one (from $k = 1$ to $k = m$). If $y_1(k) = 0$, nothing is done. If $y_1(k) = 1$, all x_i's such that $x_i(k) = 0$ and all $\overline{x_i}$'s such that $\overline{x_i(k)} = 0$ (i.e., $x_i(k) = 1$) are deleted from the conjunction respectively. Let $x_{i_1}, \ldots, x_{i_h}, \overline{x_{j_1}}, \ldots, \overline{x_{j_{h'}}}$ be the variables remained in the conjunction after testing all examples. Let $k_1, k_2, \ldots, k_{m'}$ be the indices such that $y_1(k_i) = 0$.

In the second step, we make two matrices X and Y, where X is the $m' \times h$ integer matrix defined by $X_{s,t} = x_{i_t}(k_s)$, and Y is the $m' \times h'$ integer matrix defined by $Y_{s,t} = \overline{x_{j_t}(k_s)}$. We compute the matrix product $Z = X \cdot Y^t$. Then, we count the number of elements $Z_{i,j}$ such that $Z_{i,j} = 0$. Since $(\forall k)(y_1(k) = x_{i_s}(k) \wedge \overline{x_{j_t}(k)})$ holds iff. $Z_{s,t} = 0$, the correct number is output.

Now we analyze the time complexity. Clearly, the first step takes $O(mn)$ time. The second step takes $O(m^{\omega-2}n^2 + mn^{\omega-1})$ time since $m' \leq m$, $h \leq n$ and $h' \leq n$. Therefore, the total time complexity is $O(m^{\omega-2}n^2 + mn^{\omega-1})$.

Counting of $x \oplus y$ functions

This case is easier than the above case and we do not require matrix multiplication. First note that $x \oplus y = z$ iff. $x \oplus z = y$. Let $str(x_i)$ be the sequence of Boolean values of $x_i(k)$ (i.e., $str(x_i) = \langle x_i(1), x_i(2), \ldots, x_i(m) \rangle$). Let $str(x_i)'$ be the sequence of Boolean values of $x_i(k) \oplus y_1(k)$ (i.e., $str'(x_i) = \langle x_i(1) \oplus y_1(1), x_i(2) \oplus y_1(2), \ldots, x_i(m) \oplus y_1(m) \rangle$). Then, we count the number of pairs (x_i, x_j) such that $str(x_i) = str'(x_j)$. Of course, it would take $O(mn^2)$ time if we used a naive algorithm. But, we can reduce the time complexity to $O(mn)$ by constructing a *trie* for $str(x_i)$'s and $str'(x_j)$'s, as in [4]. It is easy to modify the algorithm for counting of $\overline{x \oplus y}$ functions.

Theorem 1. *The counting problem for Boolean functions of two inputs can be solved in $O(m^{\omega-2}n^2 + mn^{\omega-1})$ time.*

Extension to $D > 2$

Using the above mentioned algorithm, we can develop an $O(m^{\omega-2}n^D + mn^{D+\omega-3})$ time algorithm for any fixed $D \geq 2$. Here, we briefly describe the method for $D = 3$. Note that

$$f(x, y, z) = (z \wedge f(x, y, 1)) \vee (\overline{z} \wedge f(x, y, 0))$$

holds for any Boolean function $f(x, y, z)$. Thus, we can count the number of Boolean functions $f(x_i, x_j, x_h)$ for fixed x_h by multiplying the number of $f_1(x_i, x_j)$'s such that $y_1(k) =$

$f_1(x_i(k), x_j(k))$ holds for examples with $x_h(k) = 1$ and the number of $f_2(x_i, x_j)$'s such that $y_1(k) = f_2(x_i(k), x_j(k))$ holds examples with $x_h(k) = 0$.

Corollary 1. *The counting problem for Boolean functions of D inputs can be solved in $O(m^{\omega-2}n^D + mn^{D+\omega-3})$ time.*

3.2 An Algorithm for Boolean Networks

In this subsection, we consider the case of $l = n$ where the technique can be applied to any l.

In addition to matrix multiplication, we use the randomized fingerprint function developed by Karp and Rabin [11, 18]. Here we briefly review the function. Let $s = \langle s_1, s_2, \ldots, s_m \rangle$ and $t = \langle t_1, t_2, \ldots, t_m \rangle$ be strings of length m over $\{0, 1\}$, respectively. Let p be a prime number. We define the fingerprint function $F_p(s)$ by

$$F_p(\langle s_1, s_2, \ldots, s_m \rangle) \;=\; s_1 \cdot 2^0 + s_2 \cdot 2^1 + \ldots + s_m \cdot 2^{m-1} \bmod p\,.$$

It was shown that, by choosing a prime number less than $\tau = \Theta(cm \log(cm))$ uniformly at random, $Prob(F_p(s) = F_p(t)) \leq \frac{1}{c}$ holds for any $s \neq t$.

For simplicity, we describe the counting algorithm for Boolean functions of the form $x_i \wedge \overline{x_j}$, where it can be easily modified for the other functions in AND and OR categories. For each y_i, we compute $F_p(\langle y_i(1), y_i(2), \ldots, y_i(m) \rangle)$. We make two matrices X and Y, where X is the $m \times n$ integer matrix defined by $X_{k,i} = x_i(k) \cdot 2^{k-1} \bmod p$, and Y is the $m \times n$ integer matrix defined by $Y_{k,j} = \overline{x_j(k)}$. Next, we compute the matrix product $Z = X \cdot Y^t$ under modulo p (i.e., under $GF(p)$). Then, we partition $Z_{i,j}$'s into groups so that each group consists of elements having the same value. For each y_h, we output the number of elements in the group that has the same value as $F_p(y_h)$.

It is easy to see that

$$Z_{i,j} = (x_i(1) \wedge \overline{x_j(1)}) \cdot 2^0 + (x_i(2) \wedge \overline{x_j(2)}) \cdot 2^1 + \ldots + (x_i(m) \wedge \overline{x_j(m)}) \cdot 2^{m-1} \bmod p\,.$$

Therefore, for any triplet (y_h, x_i, x_j) satisfying $(\forall k)(y_h(k) = x_i(k) \wedge \overline{x_j(k)})$, $F_p(y_h) = Z_{i,j}$ always holds, whereas $F_p(y_h) \neq Z_{i,j}$ holds with high probability for the other triplets (y_h, x_i, x_j). By letting $\tau = \Theta(mn^{3+\alpha} \log(mn^{3+\alpha}))$, the failure probability (i.e., the probability that a false number is output for some y_h) can be made less than $\frac{1}{n^\alpha}$.

In order to treat XOR functions, we can use the following fact: $x_i \oplus x_j = (x_i \vee x_j) - (x_i \wedge x_j)$, where we omit details here.

Now we consider the time complexity. Since we assume the standard RAM model in this paper, each arithmetic operation for $O(\log(nm))$ bit integers can be done in constant time. Therefore, we can assume that each operation in $GF(p)$ can be done in constant time. Generation of a random prime number can be done in $O(poly(\log(\tau)))$ time using a Monte-Carlo type randomized algorithm [18]. Since all known matrix multiplication algorithms are available in any ring [18], $Z = X \cdot Y^t$ can be computed in $O(m^{\omega-2}n^2 + mn^{\omega-1})$ time. Since the other parts take $O(n^2 \log n + mn)$ time, we have:

Theorem 2. *The counting problem for Boolean networks of $D = 2$ can be solved in $O(m^{\omega-2}n^2 + mn^{\omega-1})$ time with high probability.*

Using the same technique as in Corollary 1, we can extend the algorithm for any fixed D.

Corollary 2. *The counting problem for Boolean networks of fixed D can be solved in $O(m^{\omega-2}n^D + mn^{D+\omega-3})$ time with high probability.*

3.3 Allowing Errors

In practice, input data may contain noises. In such a case, Boolean functions $f(x_{i_1}, \ldots, x_{i_D})$ whose errors (i.e., $|\{k|y_j(k) \neq f(x_{i_1}, \ldots, x_{i_D})\}|$) are less than some threshold should be output. A trivial algorithm takes $O(mn^{D+1})$ time for the case of $l = n$. Matrix multiplication is also useful in order to count the error. Using matrix multiplication, we can reduce the time complexity to $O(m^{\omega-2}n^{D+1} + mn^{D+\omega-2})$.

A simple and well-known random sampling technique can also be used. Randomly choosing $\Omega(\log m)$ samples from m, we can develop an $O((\log m)n^{D+1})$ time randomized algorithm which outputs all Boolean functions with errors less than the threshold with high probability, although Boolean functions with errors slightly more than the threshold are also output. But, both algorithms are almost trivial and are not practical because each algorithm takes more than $O(n^3)$ time even for $D = 2$.

3.4 An Approximation Algorithm

As mentioned in Section 2, the identification problem is NP-hard if D is not a constant. However, we can develop a polynomial time approximation algorithm, using the technique developed for an approximation algorithm for the minimum key problem [2]. As in [2], we reduce the problem to the SET COVER problem. Recall that SET COVER is, given a collection of sets $\mathcal{S} = \{S_1, \ldots, S_M\}$ over a set U ($S_i \subseteq U$), to find a minimum cardinality set $C \subseteq \mathcal{S}$ such that $\bigcup_{S_i \in C} S_i = U$. For each of y_j's, we compute a set of variables $\{x'_{i_1}, \ldots, x'_{i_h}\}$ as follows. From given examples, we construct U and S_i's by

$$\begin{aligned} U &= \{(k, k')|\ k < k' \text{ and } y_j(k) \neq y_j(k')\}, \\ S_i &= \{(k, k')|\ x_i(k) \neq x_i(k')\} \cap U. \end{aligned}$$

Then, we apply an approximation algorithm for SET COVER [10] to S_i's and U. Since SET COVER can be approximated within a factor of $\ln|U| + 1$, we have:

Theorem 3. *Assume that $f_j(x_{i_1}(k), \ldots, x_{i_D}(k)) = y_j(k)$ holds for all k. Then a set of variables $\{x_{i'_1}, \ldots, x_{i'_h}\}$ such that $h \leq (2\ln m + 1)D$ and there exists a Boolean function f'_j satisfying $f'_j(x_{i'_1}(k), \ldots, x_{i'_h}(k)) = y_j(k)$ for all k can be found in polynomial time.*

Although a set of variables can be found in polynomial time, it seems difficult to determine f'_j in polynomial time because there exist 2^{2^d} Boolean functions with d input variables. It should be noted that description of a function needs $\Omega(2^d)$ space unless types of Boolean functions are restricted.

3.5 An Algorithm for Finding Functional Relations

Although the domain of values is restricted to $\{0, 1\}$ in Boolean networks, the algorithm in Section 3.2 can be extended for other fixed size domains. As in Section 3.2, we explain the algorithm for the case of $D = 2$. Extension to other D's can be done as in Section 3.1.

Let Σ be the domain (i.e., $x_i(k), y_h(k) \in \Sigma$), where we let $b = |\Sigma|$. In this case, we use the fingerprint function on base b:

$$F_{p,b}(\langle s_1, s_2, \ldots, s_m\rangle) = s_1 \cdot b^0 + s_2 \cdot b^1 + \ldots + s_m \cdot b^{m-1} \bmod p,$$

where it is known that a property similar to that of F_p holds for this function [18]. For each function f in $\Sigma \times \Sigma \to \Sigma$, we examine whether or not there exists a triplet (y_h, x_i, x_j) such that

$y_h(k) = f(x_i(k), x_j(k))$ holds for all k. For each $\alpha \in \Sigma$, let X^α be the $m \times n$ matrix defined by

$$X^\alpha_{k,i} = \begin{cases} 1, & \text{if } x_i(k) = \alpha, \\ 0, & \text{otherwise.} \end{cases}$$

For each $\alpha \in \Sigma$, let Y^α be the $m \times n$ matrix defined by $Y^\alpha_{k,j} = f(\alpha, x_j(k))$, where we encode each element in Σ by using an element in $\{0, 1, \ldots, b-1\}$. Let $Z = \sum_\alpha X^\alpha \cdot (Y^\alpha)^t$. Then, $F_{p,b}(\langle y_h(1), \ldots, y_h(m)\rangle) = Z_{i,j}$ holds if $y_h(k) = f(x_i(k), x_j(k))$ holds for all k, and $F_{p,b}(\langle y_h(1), \ldots, y_h(m)\rangle) \neq Z_{i,j}$ holds with high probability if $y_h(k) \neq f(x_i(k), x_j(k))$ holds for some k.

Theorem 4. *For a fixed domain, the number of functional relations $y_j = f(x_{i_1}, \ldots, x_{i_D})$ can be computed for all y_j in $O(m^{\omega-2}n^D + mn^{D+\omega-3})$ time for fixed D with high probability.*

4 Concluding Remarks

In this paper, we presented improved algorithms for the identification of the Boolean networks and related problems. Note that if an ultimate matrix multiplication algorithm ($\omega = 2$?) were developed, the time complexity of the identification algorithm for a Boolean network of $D = 2$ would be $O(n^2 \log n + mn)$, which is nearly optimal in the case of $m \geq n$. However, it is still far from optimal when $m \ll n$. Therefore, development of faster algorithms, in particular, development of an algorithm for which the exponent of n is less than 2 (for $D = 2$) is an open problem. In the identification of functional relations, we assumed a fixed domain. Development of an $o(mn^D)$ time algorithm for any domain is an open problem.

Although we developed improved algorithms, they are not practical because of the following reasons: fast matrix multiplication algorithms are not practical; most of the proposed algorithms are not robust for noises; too simplified models are assumed. Development of practical algorithms is important future work.

Acknowledgments

This work was supported in part by a Grant-in-Aid for Scientific Priority Areas, "Genome Science," from the Ministry of Education, Science, Sports and Culture in Japan.

References

[1] A.V. Aho, Algorithms for finding patterns in strings, in J. Van Leeuwen (ed.) *Handbook of Theoretical Computer Science* Vol. A, 1990.

[2] T. Akutsu and F. Bao, Approximating minimum keys and optimal substructure screens, *Proc. 2nd Int. Conf. Computing and Combinatorics*, 290–299, 1996.

[3] T. Akutsu, S. Miyano and S. Kuhara, Identification of genetic networks from a small number of gene expression patterns under the Boolean network model, *Proc. Pacific Symposium on Biocomputing '99* (PSB'99), 17–28, 1999.

[4] T. Akutsu, S. Miyano and S. Kuhara, Fast identification of Boolean networks, Technical Report 99-AL-66, Information Processing Society of Japan, 25–32, 1999.

[5] A. Amir and M. Farach, Efficient two-dimensional approximate matching of non-rectangular figures, *Proc. ACM-SIAM Symp. on Discrete Algorithms*, 212-223, 1991.

[6] D. Coppersmith and S. Winograd, Matrix multiplication via arithmetic progression, *J. Symbolic Computation* **9**, 251–280, 1990.

[7] P. D'haeseleer, X. Wen, S. Fuhrman and R. Somogyi, Linear modeling of mRNA expression levels during CNS development and injury, *Proc. Pacific Symp. Biocomputing'99* (PSB99), 41–52 (1999).

[8] M.J. Fisher and M.S. Paterson, String matching and other products, *Proc. SIAM-AMS Conference on Complexity of Computation*, AMS, 113–125, 1974.

[9] X. Huang and V. Pan, Fast rectangular matrix multiplication, *Proc. ACM Symp. Parallel Algebraic and Symbolic Computation*, 11–23, 1997.

[10] D.S. Johnson, Approximation algorithms for combinatorial problems, *J. Computer and System Sciences* **9**, 256–278, 1974.

[11] R.M. Karp and M.O. Rabin, Efficient randomized pattern-matching algorithms, *IBM Journal of Research and Development* **31**, 249–260, 1985.

[12] M.J. Kearns and U.V. Vazirani, *An Introduction to Computational Learning Theory*, The MIT Press, 1994.

[13] S. Liang, S. Fuhrman and R. Somogyi, REVEAL, a general reverse engineering algorithm for inference of genetic network architectures, *Proc. Pacific Symposium on Biocomputing '98* (PSB'98), 18–29, 1998.

[14] N. Littlestone, Learning quickly when irrelevant attributes abound: a new linear-threshold algorithm, *Machine Learning* **2**, 285–318, 1988.

[15] K. Makino, *Studies on positive and horn Boolean functions with applications to data analysis*, Ph.D Thesis, Kyoto University, 1997.

[16] H. Mannila and K. Räihä, Dependency inference, *Proc. 13th VLDB Conference*, 155–158, 1987.

[17] H. Mannila and K. Räihä, On the complexity of inferring functional dependencies, *Discrete Applied Mathematics* **40**, 237–243, 1992.

[18] R. Motowani and P. Raghavan, *Randomized Algorithms*, Cambridge Univ. Press, 1994.

[19] L.G. Valiant, A theory of the learnable, *Communications of the ACM* **27**, 1134–1142, 1984.