Practical Algorithms for Pattern Based Linear Regression

Hideo Bannai[1], Kohei Hatano[1], Shunsuke Inenaga[1,2], and Masayuki Takeda[1,3]

[1] Department of Informatics, Kyushu University,
6-10-1 Hakozaki, Higashi-ku, Fukuoka 812-8581, Japan
{bannai, hatano, shunsuke.inenaga, takeda}@i.kyushu-u.ac.jp
[2] Japan Society for the Promotion of Science
[3] SORST, Japan Science and Technology Agency (JST)

Abstract. We consider the problem of discovering the optimal pattern from a set of strings and associated numeric attribute values. The goodness of a pattern is measured by the correlation between the number of occurrences of the pattern in each string, and the numeric attribute value assigned to the string. We present two algorithms based on suffix trees, that can find the optimal substring pattern in $O(Nn)$ and $O(N^2)$ time, respectively, where n is the number of strings and N is their total length. We further present a general branch and bound strategy that can be used when considering more complex pattern classes. We also show that combining the $O(N^2)$ algorithm and the branch and bound heuristic increases the efficiency of the algorithm considerably.

1 Introduction

Fundamental biological molecules such as DNA, RNA, and proteins can be regarded as strings over a certain alphabet. Although the whole genomic sequences of many species are now becoming available, there is still much that is unknown about the information that lie hidden in them. Computational analysis of these sequences rely on the principle that similarity as strings implies similarity in their sequence structure, which in turn implies similarity in their functions. Therefore, methods for efficiently and effectively discovering meaningful *patterns*, or sequence elements which are conserved in a given set of strings, is an important problem in the field of Bioinformatics [1].

Earlier work on pattern discovery focus on discovering the most conserved pattern in a given set of strings, generally preferring *longer* patterns which occur in *most* of the sequences in the set. Another situation is when we are given two sets of strings, where one set (positive set) consists of sequences known to possess some biological characteristic, while the other (negative set) consists of sequences known not to posses these characteristics. The problem is to find a discriminating pattern, that is, a pattern which occurs in most strings of the positive set, but does *not* occur in most of the strings of the negative set [2–6].

Recently, there have been several works which incorporate numeric attributes which are obtained from other sources, e.g. gene expression data obtained from

microarray experiments, in order to find meaningful patterns more effectively [7–11]. The basic idea of these methods is to find sequences elements whose occurrences in the sequences are correlated with the numeric attributes. For example, gene expression is regulated by molecules called *transcription factors*, which bind to specific sequences usually in the upstream of the coding region of a gene. The binding sites for a given transcription factor are fairly conserved across genes which are regulated by the same transcription factor. Therefore, if we can find sequence elements which occur in upstream regions of genes which are relatively highly expressed, while not occurring in upstream regions of genes whose expression is relatively low (or vice versa), such patterns are likely to be binding sites of specific transcription factors.

In [7], substring patterns of up to length 7 are scored according to the linear fit between the number of occurrences in the upstream region and the expression level of the gene. However, they do not consider any algorithm for solving the problem efficiently. Also, the choice of the maximum pattern length is arbitrary and it is not guaranteed that the optimal pattern will be found. Algorithmic work for solving a similar problem has been considered in [8,9]. In this problem setting, the number of occurrences of a pattern is only considered as an indicator value of 0 or 1, i.e. whether the pattern occurs in the string or not. Based on the algorithm for solving the color set size problem [12], a very efficient $O(N)$ time algorithm for finding the optimal substring pattern in this problem setting is given in [9]. A general branch and bound strategy that can be used for more complex patterns where the problem can be NP-hard, is given in [8].

Although the algorithms above have been shown to discover similar motifs as in [7], it is generally believed that multiple occurrences of a binding site motif in the upstream region of a gene will strengthen the function of the transcription factor for that gene. In this paper, we give an efficient algorithm to discover the optimal pattern, taking into account the number of occurrences of the pattern in each string, as in the problem setting in [7]. We first present two simple algorithms based on the suffix tree data structure that finds the optimal substring pattern (without a restriction in the length of the pattern), respectively in $O(nN)$ and $O(N^2)$ time. We further develop and apply a branch and bound strategy in order to speed up the algorithm, also allowing the problem to be solved for more complex and descriptive classes of patterns. The algorithms developed are applied to real biological data to show the efficiency and effectiveness of the approach.

2 Preliminaries

Let Σ be a finite alphabet. An element of Σ^* is called a *string*. Strings x, y, and z are said to be a *prefix*, *substring*, and *suffix* of string $w = xyz$, respectively. The length of a string w is denoted by $len(w)$. The empty string is denoted by ε, that is, $len(\varepsilon) = 0$. For any set S, let $|S|$ denote the cardinality of the set. The empty set is denoted by $\emptyset$, that is, $|\emptyset| = 0$. Let $\mathbf{R}$ represent the set of real numbers.

Let Π be a set of *patterns*. We call a function defined over a text string and pattern $\psi : \Sigma^* \times \Pi \rightarrow \mathbf{R}$ a *matching function*. Let $\psi_p : \Sigma^* \rightarrow \mathbf{R}$ represent the matching function for a fixed $p \in \Pi$, that is, $\psi(s, p) = \psi_p(s)$ for any text string $s \in \Sigma^*$. For the matching function value, we shall consider the number of occurrences of a given pattern in the text string. A *substring pattern* p is a pattern $p \in \Pi = \Sigma^*$, where the matching function value $\psi_p(s)$ is defined as the number of substrings in s which is equal to p. A *don't care pattern* p is a pattern $p \in \Pi = (\{.\} \cup \Sigma)^*$, where "." is a don't care symbol, and the matching function value $\psi_p(s)$ is defined as the number of substrings in s which can be obtained from p by appropriate substitution of the don't care symbols with characters of the alphabet Σ. For the above two pattern classes, we shall refer to Σ or $\{.\} \cup \Sigma$ as the *pattern alphabet*.

For the rest of the paper, we assume that we are given as input, a sequence of ordered pairs consisting of a string and an associated numeric attribute value: $\{(s_1, y_1), \ldots, (s_n, y_n)\} \subset \Sigma^* \times \mathbf{R}$. Let $N = \sum_{i=1}^{n} len(s_i)$ represent the total length of the input strings. We denote by $\boldsymbol{y} = (y_1, \ldots, y_n)^T \in \mathbf{R}^n$, the vector consisting of the numeric attribute values. Further, for a given pattern p, we denote by $\boldsymbol{\psi_p(s)} = (\psi_p(s_1), \ldots, \psi_p(s_n))^T \in \mathbf{R}^n$, the vector consisting of the matching function values for the input text strings. We define for later use, a preorder over patterns as follows:

Definition 1. *For any $p', p \in \Pi$, denote $p' \preceq p$ if for all $\{s_i \mid i = 1, \ldots, n\}$, $\psi(s_i, p') \leq \psi(s_i, p)$.*

We now consider how to score the *goodness* of a given pattern. For a given $\boldsymbol{x} = (x_1, \ldots, x_n)^T \in \mathbf{R}^n$, let $\boldsymbol{X} = (\mathbf{1}, \boldsymbol{x})$, and define

$$RSS(\boldsymbol{y}|\boldsymbol{x}) = ||\boldsymbol{y} - \boldsymbol{X}\hat{\boldsymbol{\beta}}||^2 = \sum_{i=1}^{n} (y_i - (\hat{\beta}_0 + \hat{\beta}_1 x_i))^2$$

where $\hat{\boldsymbol{\beta}} = (\hat{\beta}_0, \hat{\beta}_1)^T$ are the least square estimates of $\boldsymbol{\beta}$ in the linear model

$$\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}.$$

We consider the problem of finding the pattern which can best fit the numeric attribute values y_i with respect to $\psi_p(s_i)$.

Definition 2 (Pattern Based Linear Regression). *We define the pattern based linear regression problem as follows. Given $\{(s_1, y_1), \ldots, (s_n, y_n)\} \subset \Sigma^* \times \mathbf{R}$, and a matching function ψ, find the pattern $p \in \Pi$ that minimizes*

$$RSS(\boldsymbol{y}|\boldsymbol{\psi_p(s)}) = ||\boldsymbol{y} - (\mathbf{1}, \boldsymbol{\psi_p(s)})\hat{\boldsymbol{\beta}}||^2 = \sum_{i=1}^{n} (y_i - (\hat{\beta}_0 + \hat{\beta}_1 \psi_p(s_i)))^2$$

where $\hat{\boldsymbol{\beta}} = (\hat{\beta}_0, \hat{\beta}_1)^T$ are the least square estimates of $\boldsymbol{\beta}$ in the linear model

$$\boldsymbol{y} = (\mathbf{1}, \boldsymbol{\psi_p(s)})\boldsymbol{\beta} + \boldsymbol{\epsilon}.$$

We note that the consistency problem [13,14] is a special case of our problem. Since the consistency problem is shown to be NP-complete for several pattern classes (e.g. *subsequence patterns*), the above problem is NP-hard for such cases. An exception is the case for the substring pattern class, for which we shall present efficient solutions in Section 3.

3 Methods

3.1 Finding the Optimal Substring Pattern

When considering substring patterns, it can be shown that the number of possible patterns which give distinct *RSS* scores is linear in the total length of strings, i.e. $O(N)$. We make use of a very convenient and well studied data structure called *suffix trees*.

Generalized Suffix Trees A *suffix tree* [15] for a given string s is a rooted tree whose edges are labeled with substrings of s, satisfying the following characteristics. For any node v in the suffix tree, let $l(v)$ denote the string spelled out by concatenating the edge labels on the path from the root to v. For each leaf node v, $l(v)$ is a distinct suffix of s, and for each suffix of s, there exists such a leaf v. Furthermore, each node has at least two children, and the first character of the labels on the edges to its children are distinct. A generalized suffix tree (GST) for a set of n strings $S = \{s_1, \ldots, s_n\}$ is basically a suffix tree for the string $s_1\$_1 \cdots s_n\$_n$, where each $\$_i$ $(1 \leq i \leq n)$ is a distinct character which does not appear in any of the strings in the set. However, all paths are ended at the first appearance of any $\$_i$, and each leaf is labeled with id_i. An example of a GST is shown in Fig. 1. It is well known that suffix trees (and generalized suffix trees) can be represented in linear space and constructed in linear time [15] with respect to the length of the string (total length of the strings for GST).

Notice that candidate substring patterns may be restricted to those represented by nodes of the generalized suffix tree. This is because, for any substring pattern that does not correspond to a path in the suffix tree, the pattern does not occur in any of the strings in the set. Also, note that for a given pattern that does correspond to a path in the suffix tree, all occurrences of the pattern in the strings are represented by the leaves of the suffix tree in the subtree below this path. This means that for any substring pattern that corresponds to a path that ends in the middle of an edge of the suffix tree, its occurrences in the strings are identical to the occurrences of the substring pattern corresponding to the path extended to the next node.

As stated in the Introduction, the pattern based linear regression problem has been shown to be solvable in $O(N)$ time if the matching function is considered to be an indicator function returning the value 0 or 1 [9]. However, it is assumed that the score of a pattern is a function of the sum of the matching function values and the sum of the numeric attribute values of the strings that the pattern occurs in. The algorithm cannot be applied to our case since we

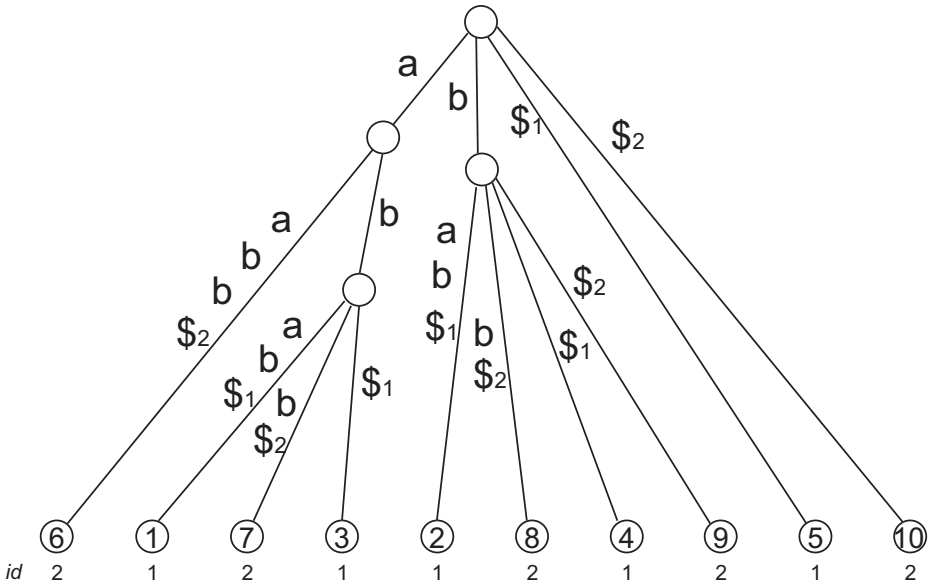

Fig. 1. Example of a generalized suffix tree for the string set { `abab`, `aabb` }.

require the matching function values of each string in order to compute the *RSS* score. Below, we give two algorithms which calculate the score for each candidate pattern p corresponding to a node in the generalized suffix tree.

An $O(N^2)$ Algorithm Calculating the score for each of the $O(N)$ candidate patterns requires the calculation of $\boldsymbol{\psi_p(s)}$ for each p, as well as the calculation of the least square estimates. The former can be calculated in $O(N)$ time using well known linear time string pattern matching algorithms such as the Knuth-Morris-Pratt algorithm [16]. The latter can be calculated by first obtaining the least square estimate of the parameters:

$$\hat{\boldsymbol{\beta}} = (\boldsymbol{X^T X})^{-1} \boldsymbol{X^T y},$$

where $\boldsymbol{X} = (\mathbf{1}, \boldsymbol{\psi_p(s)})$. It is not difficult to see that this can be calculated in $O(n)$ time. Further, $RSS(\boldsymbol{y}|\boldsymbol{\psi_p(s)})$ can be calculated in $O(n)$ time, and therefore the resulting time complexity is $O(N) \cdot O(N+n) = O(N^2)$.

An $O(Nn)$ Algorithm Consider assigning a vector of length n at each node and leaf of the suffix tree. The vectors are initialized as follows: for an internal node, all values are set to 0. For a leaf labeled with id_i, the value at the ith position is set to 1, and the rest is set to 0. Then, with a bottom-up (postorder) traversal on the suffix tree, we add up the values in the vector at each node,

element-wise, into the vector of its parent node. The result is that we obtain, at each node, a vector of length n where each position i of the vector represents the number of leaves in the subtree of the suffix tree, with id_i. This corresponds to the number of times the substring pattern occurs in string s_i, and consequently, the vector represents $\boldsymbol{\psi_p(s)}$. This means that $\boldsymbol{\psi_p(s)}$ can be calculated in $O(n)$ time for each pattern, resulting in a total of $O(Nn)$ time for the score calculations.

3.2 Branch and Bound Strategy

Since the pattern based linear regression problem can be NP-hard when considering more complex pattern classes, we propose an enumerative branch and bound framework for finding the optimal pattern. The basic idea of the enumeration is similar to previous works [2–6, 8]. The main contrivance of this paper is in the method for calculating the lower bound of the *RSS* score for specific subspaces of the pattern space.

The algorithm proceeds by traversing and enumerating nodes on a *search tree*, where each node in the tree represents some pattern in Π. For any pattern p in the search tree, let p' be a pattern represented by a node in the subtree rooted at the node for p. While traversing the search tree at the node corresponding to p, suppose that we are able to calculate a lower bound for the *RSS* for any pattern p'. If this lower bound is greater than the current best *RSS* found in the traversal, we know that the score for p' cannot be below the current best *RSS*. This allows us to prune the search space by disregarding the subtree of the search tree rooted at the node corresponding to p.

Below, we show how such a lower bound can be calculated. The assumptions for our calculations below is that $p' \preceq p$. For the case of string patterns and don't care patterns, this assumption can be fulfilled by considering the search tree described as follows. The root corresponds to the empty string ε, and each node v will have child nodes whose pattern corresponds to the pattern obtained by extending the pattern at node v by one character of the pattern alphabet. Although we do not elaborate in this paper, the same branch and bound approach can be applied to a variety of other patterns such as approximate patterns and degenerate patterns.

Problem 1 (lower bound of the residual sum of squares for $p' \preceq p$). Given some pattern $p \in \Pi$, find a lower bound of the score function $RSS(\boldsymbol{y}|\boldsymbol{\psi_{p'}(s)})$ for any pattern $p' \preceq p$.

Below, let $\boldsymbol{x} = (x_1, \ldots, x_n)^T = \boldsymbol{\psi_p(s)}$. Also, let $\boldsymbol{x}^{\mathrm{opt}} = (x_1^{\mathrm{opt}}, \ldots, x_n^{\mathrm{opt}})^T = \arg\min_{\boldsymbol{z} \in D_x} RSS(\boldsymbol{y}|\boldsymbol{z})$ where $D_x = \{(z_1, \ldots z_n)^T \mid 0 \leq z_i \leq x_i,\ i = 1, \ldots, n\}$, and let $\hat{\boldsymbol{\beta}}^{\mathbf{opt}} = (\hat{\beta}_0^{\mathrm{opt}}, \hat{\beta}_1^{\mathrm{opt}})$ be the least square estimates for the linear model $\boldsymbol{y} = (\mathbf{1}, \boldsymbol{x}^{\mathrm{opt}})\boldsymbol{\beta} + \boldsymbol{\epsilon}$. Our objective now is to find $RSS(\boldsymbol{y}|\boldsymbol{x}^{\mathrm{opt}})$. This can be considered as a relaxed version of the problem stated above, since we do not require that there exists a pattern $p' \preceq p$ such that $\boldsymbol{\psi_{p'}(s)} = \boldsymbol{x}^{\mathrm{opt}}$. The following theorem gives a simple lower bound on the *RSS* score.

Algorithm 1: Simple algorithm for calculating a lower bound of *RSS*.

Input: $\boldsymbol{x} = (x_1, \ldots, x_n)^T$,$\boldsymbol{y} = (y_1, \ldots, y_n)^T$
Output: $\mathsf{lb} \leq \min\{RSS(\boldsymbol{y}|\boldsymbol{z}) \mid \boldsymbol{z} = (z_1, \ldots, z_n)^T, 0 \leq x'_i \leq x_i\}$

```
1 X ← {i | x_i = 0};
2 if X = ∅ then return 0;        /* Can move all x_i onto a single line. */
3 m ← Σ_{i∈X} y_i/|X|;                    /* mean of values {y_i | x_i = 0} */
4 lb = Σ_{i∈X} (m − x_i)^2;   /* residual sum of squares for points {x_i = 0} */
5 return lb
```

Theorem 1 (simple lower bound). *Let $X = \{i \mid x_i = 0\}$. If $X \neq \emptyset$, then*

$$\sum_{i \in X} (m - y_i)^2 \leq RSS(\boldsymbol{y}|\boldsymbol{x}^{\text{opt}})$$

where $m = \sum_{i \in X} y_i/|X|$.

Proof. For any i, if $x_i = 0$, then since $0 \leq x_i^{\text{opt}} \leq x_i$ we have that $x_i^{\text{opt}} = 0$. Ignoring the residuals for all data points $x \neq 0$, the minimum possible residual sum of squares for data where $x_i = x_i^{\text{opt}} = 0$ must be smaller than the residual sum of squares for the entire data set. □

The simple lower bound can be calculated with the pseudo-code shown in Algorithm 1.

Next, we try to improve on this lower bound. The following lemma gives the conditions between $\boldsymbol{x}^{\text{opt}}$ and the regression line.

Lemma 1. *For all $i = 1, \ldots, n$, if $\hat{\beta}_1^{\text{opt}} > 0$ then*

$$x_i^{\text{opt}} = \begin{cases} 0 & \text{if } x_i = 0 \text{ or } y_i \leq \hat{\beta}_0^{\text{opt}} \\ (y_i - \hat{\beta}_0^{\text{opt}})/\hat{\beta}_1^{\text{opt}} & \text{otherwise } (x_i > 0 \text{ and } y_i > \hat{\beta}_0^{\text{opt}}) \end{cases}$$

and if $\hat{\beta}_1^{\text{opt}} < 0$,

$$x_i^{\text{opt}} = \begin{cases} 0 & \text{if } x_i = 0 \text{ or } y_i \geq \hat{\beta}_0^{\text{opt}} \\ (y_i - \hat{\beta}_0^{\text{opt}})/\hat{\beta}_1^{\text{opt}} & \text{otherwise } (x_i > 0 \text{ and } y_i < \hat{\beta}_0^{\text{opt}}). \end{cases}$$

Proof. We will prove the case for $\hat{\beta}_1^{\text{opt}} > 0$. The case for $\hat{\beta}_1^{\text{opt}} < 0$ can be done similarly. The lemma states that the points $(x_1^{\text{opt}}, y_1), \ldots, (x_n^{\text{opt}}, y_n)$ lie either on the regression line $y = \hat{\beta}_0^{\text{opt}} + \hat{\beta}_1^{\text{opt}} x$, or on the y-axis. Suppose there exist points (x_i^{opt}, y_i) to the contrary, that is, $x_i^{\text{opt}} > 0$ and the point is not on the regression line. If $y_i < \hat{\beta}_0^{\text{opt}}$, then since $\hat{\beta}_1^{\text{opt}} > 0$, considering point $(0, y_i)$ instead of (x_i^{opt}, y_i) would give a smaller residual, contradicting the definition of $\boldsymbol{x}^{\text{opt}}$. If $y_i > \hat{\beta}_0^{\text{opt}}$, then again since $\hat{\beta}_1^{\text{opt}} > 0$, (x_i^{opt}, y_i) cannot lie to the right of the regression line, or we can replace (x_i^{opt}, y_i) with a point $((y_i - \hat{\beta}_0^{\text{opt}})/\hat{\beta}_1^{\text{opt}}, y_i)$ on the regression line with a smaller residual. We can also say that the points

cannot lie *left* of the regression line, since we can construct a new regression line passing the point $(0, \beta_0)$ that lies left of the points, and replace all points (x_i^{opt}, y_i) with points on the new regression line. The residual of the new points are clearly smaller, and again contradicts the definition of $\boldsymbol{x}^{\text{opt}}$. □

Corollary 1. *Let* $X^{\text{opt}} = \{i \mid x_i^{\text{opt}} = 0\}$. *If* $X^{\text{opt}} = \emptyset$, $RSS(\boldsymbol{y}|\boldsymbol{x}^{\text{opt}}) = 0$. *If* $X^{\text{opt}} \neq \emptyset$, *then*

$$\hat{\beta}_0^{\text{opt}} = \sum_{i \in X} y_i / |X^{\text{opt}}|$$

and

$$RSS(\boldsymbol{y}|\boldsymbol{x}^{\text{opt}}) = \sum_{i \in X^{\text{opt}}} (\hat{\beta}_0^{\text{opt}} - x_i^{\text{opt}})^2.$$

Proof. As a consequence of Lemma 1. If $X = \emptyset$, then all points (x_i^{opt}, y_i) lie on the regression line. Otherwise, since the residual for all points $x_i^{\text{opt}} > 0$ is 0, β_0 is chosen to minimize the residual sum of squares of points where $x_i^{\text{opt}} = 0$. □

In order to calculate the lower bound, the problem now is how to obtain the value $\hat{\beta}_0^{\text{opt}}$. Although the exact value of $\hat{\beta}_0^{\text{opt}}$ is not known beforehand, Algorithm 2 shows how to calculate the minimum residual sum of squares for any $\boldsymbol{z} = (z_1, \ldots, z_n)^T$ satisfying the constraint: $0 \leq z_i \leq x_i$ for all $i = 1, \ldots, n$.

Corollary 2. *Let*

$$X_+^{\text{opt}} = \{i \in X^{\text{opt}} \mid y_i > \hat{\beta}_0^{\text{opt}}\}$$
$$X_-^{\text{opt}} = \{i \in X^{\text{opt}} \mid y_i \leq \hat{\beta}_0^{\text{opt}}\}.$$

If $\hat{\beta}_1^{\text{opt}} > 0$, *then*

$$X_+^{\text{opt}} = \{i \mid x_i = 0, y_i > \hat{\beta}_0^{\text{opt}}\}$$
$$X_-^{\text{opt}} = \{i \mid y_i \leq \hat{\beta}_0^{\text{opt}}\}.$$

If $\hat{\beta}_1^{\text{opt}} < 0$, *then*

$$X_+^{\text{opt}} = \{i \mid x_i = 0, y_i < \hat{\beta}_0^{\text{opt}}\}$$
$$X_-^{\text{opt}} = \{i \mid y_i \geq \hat{\beta}_0^{\text{opt}}\}.$$

Proof. As a consequence of Lemma 1. When $\hat{\beta}_1^{\text{opt}} > 0$ and $y_i > \hat{\beta}_0^{\text{opt}}$, $x_i^{\text{opt}} = 0$ if and only if $x_i = 0$. Similarly, when $\hat{\beta}_1^{\text{opt}} < 0$ and $y_i < \hat{\beta}_0^{\text{opt}}$, $x_i^{\text{opt}} = 0$ if and only if $x_i = 0$. □

Theorem 2. *Algorithm 2 correctly outputs* $RSS(\boldsymbol{y}|\boldsymbol{x}^{\text{opt}})$.

Algorithm 2: Algorithm for calculating the lower bound of RSS.

Input: $\boldsymbol{x} = (x_1, \ldots, x_n)^T$, $\boldsymbol{y} = (y_1, \ldots, y_n)^T$
Output: $\min\{RSS(\boldsymbol{y}|\boldsymbol{z}) \mid \boldsymbol{z} = (z_1, \ldots, z_n)^T, 0 \leq x'_i \leq x_i\}$

1 $X \leftarrow \{i \mid x_i = 0\}$; $k \leftarrow 1$;
2 **if** $X = \emptyset$ **then return** 0; `/* Can move all` x_i `onto a single line. */`
`// assuming` $\hat{\beta}_1^{\text{opt}} > 0$ `//////////////////////////////////////`
3 $\mathsf{m} \leftarrow \sum_{i \in X} y_i / |X|$; `/* mean of values` $\{y_i \mid x_i = 0\}$ `*/`
4 $(i_1, \ldots, i_{n-|X|}) \leftarrow$ indices sorted in increasing order of $\{y_i \mid i \notin X\}$;
5 **while** $y_{i_k} \leq \mathsf{m}$ **do**
6 $\quad X \leftarrow X \cup \{i_k\}$; `/*` $x_{i_k} \to 0$ `*/`
7 $\quad \mathsf{m} \leftarrow \sum_{i \in X} y_i / |X|$; `/* update mean */`
8 $\quad k \leftarrow k + 1$;
9 **endw**
10 $\mathsf{lb} = \sum_{i \in X} (\mathsf{m} - x_i)^2$; /* $\mathsf{m} = \hat{\beta}_0^{\text{opt}}$ and $X = X^{\text{opt}}$ if $\hat{\beta}_1^{\text{opt}} > 0$ */
`// assuming` $\hat{\beta}_1^{\text{opt}} < 0$ `//////////////////////////////////////`
11 $X \leftarrow \{i \mid x_i = 0\}$; $k \leftarrow 1$;
12 $\mathsf{m} \leftarrow \sum_{i \in X} y_i / |X|$; `/* mean of values` $\{y_i \mid x_i = 0\}$ `*/`
13 $(i_1, \ldots, i_{n-|X|}) \leftarrow$ indices sorted in decreasing order of $\{y_i \mid i \notin X\}$;
14 **while** $y_{i_k} \geq \mathsf{m}$ **do**
15 $\quad X \leftarrow X \cup \{i_k\}$; `/*` $x_{i_k} \to 0$ `*/`
16 $\quad \mathsf{m} \leftarrow \sum_{i \in X} y_i / |X|$; `/* update mean */`
17 $\quad k \leftarrow k + 1$;
18 **endw**
19 **return** $\min\{\mathsf{lb}, \sum_{i \in X} (\mathsf{m} - x_i)^2\}$; /* $\mathsf{m} = \hat{\beta}_0^{\text{opt}}$ and $X = X^{\text{opt}}$ if $\hat{\beta}_1^{\text{opt}} < 0$ */

Proof. Consider the case where $\{i \mid x_i^{\text{opt}} = 0\} \neq \emptyset$ and $\hat{\beta}_1^{\text{opt}} > 0$. The claim is that $\mathsf{m} = \hat{\beta}_0^{\text{opt}}$, and $X = X^{\text{opt}}$, after the **while** loop of lines 5–9. If this can be proved, the result follows from Corollary 1. Let us split the set X thus calculated into two disjoint sets, $X_+ = \{i \in X \mid y_i > \mathsf{m}\}$ and $X_- = \{i \in X \mid y_i \leq \mathsf{m}\}$. Since the algorithm does not add indices i where $y_i > \mathsf{m}$ to X, we have that $X_+ = \{i \mid x_i = 0, y_i > \mathsf{m}\}$ from the initial construction of X at line 1. Also, since all indices i where $y_i \leq \mathsf{m}$ are added to X, we have that $X_- = \{i \mid y_i \leq \mathsf{m}\}$. Suppose $\mathsf{m} < \hat{\beta}_0^{\text{opt}}$. From Corollary 2, we have that $X_+ \supseteq X_+^{\text{opt}}$, and $X_- \subseteq \{i \mid y_i \leq \hat{\beta}_0^{\text{opt}}\} = X_-^{\text{opt}}$. However, this contradicts the assumption, since $\sum_{i \in X} y_i / |X| = \mathsf{m} \geq \hat{\beta}_0^{\text{opt}} = \sum_{i \in X^{\text{opt}}} y_i / |X^{\text{opt}}|$. Next, suppose $\mathsf{m} > \hat{\beta}_0^{\text{opt}}$. We have the opposite situation and $X_+ \subseteq X_+^{\text{opt}}$ and $X_- \supseteq X_-^{\text{opt}}$. However, due to the way X is constructed, there must have been a point in the while loop (lines 5–9) where all indices i where $y_i \leq \hat{\beta}_0^{\text{opt}}$ are added to X, and no index i where $y_i > \hat{\beta}_0^{\text{opt}}$ is added to X. From Corollary 2, at such point, $X_+ = X_+^{\text{opt}}$ and $X_- = X_-^{\text{opt}}$, which implies that $\mathsf{m} = \hat{\beta}_0^{\text{opt}}$. Due to the condition of the while loop, no other index i where $y_i > \hat{\beta}_0^{\text{opt}}$ could have been added to X afterwards, contradicting the assumption $\mathsf{m} > \hat{\beta}_0^{\text{opt}}$. Therefore, $\mathsf{m} = \hat{\beta}_0^{\text{opt}}$, and consequently $X = X^{\text{opt}}$. □

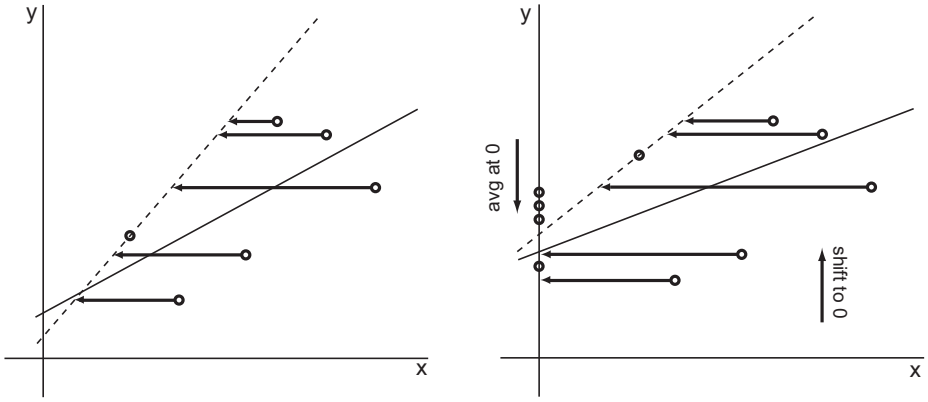

Fig. 2. Case of Algorithm 2 where $X = \emptyset$ (left) and $X \neq \emptyset$, $\beta_1 > 0$ (right).

Fig. 2 shows the basic idea of Algorithm 2. The time complexities of Algorithm 1 and 2 are both $O(n)$, since they just conduct a constant number of scans on the data set, provided that the data is initially sorted and ranked according to y_i, so that the sorting at lines 4 and 13 of Algorithm 2 can be computed in $O(n)$ time.

Combining Suffix Tree Traversal and Branch and Bound It is easy to see that for substring patterns, the generalized suffix tree itself can be used as the search tree, and we can combine the $O(N^2)$ algorithm and the pruning strategy described in this section. Combining the $O(Nn)$ algorithm and the pruning strategy is not readily realizable, since the direction of the traversal over the search tree is in the *reverse* direction. We discuss this issue further in Section 5.

4 Computational Experiments

We implement our algorithms using the C++ language, and measure the running times of our algorithm using a Sun Fire 15K (UltraSPARC III Cu 1.2GHz x 96 CPUs) using a single processor for each run. We note that the suffix tree algorithm is simulated by a suffix array structure, using the method presented in [17, 18].

For the numeric attribute values, we use the *S. cerevisiae* gene expression data obtained from microarray experiments given in [19]. The data consists of normalized log expression level ratios of genes at specific time points of the yeast cell cycle. For the string data, we use the 600 nucleotides from the upstream of the start codon of each gene.

Table 1 shows the running times of the algorithms for finding the optimal substring pattern applied to the expression data of the 14-minute time point in

Table 1. Comparison of running times of algorithm for finding optimal substring patterns from the 14-minute time point in the α-synchronized cell-cycle microarray experiment [19].

n	$O(Nn)$	$O(N^2)$	$O(N^2)$+ simple bb	$O(N^2)$+bb
100	05.96s	12.43s	05.90s	05.71s
500	00m37s	01m47s	00m30s	00m26s
1000	02m03s	05m28s	01m02s	00m53s
1500	04m09s	11m16s	01m35s	01m19s
2000	07m53s	19m55s	02m00s	01m38s
2500	12m30s	30m05s	02m35s	02m07s
3000	18m20s	42m17s	03m29s	02m51s
3500	25m17s	56m34s	04m11s	03m23s
4000	33m08s	73m06s	04m52s	03m55s
4500	42m48s	92m16s	05m34s	04m28s
5000	55m03s	113m20s	06m13s	04m59s
5500	67m01s	136m25s	07m08s	05m41s
5907	75m55s	157m27s	07m57s	06m21s

the α-synchronized cell-cycle microarray experiment. The times are measured for various sizes of n by a random sampling from the entire set of 5907 genes which were available for this time point. Note that since all strings are of fixed length, $N = 600n$. From the table, we see that the $O(Nn)$ and $O(N^2)$ time algorithms are able to find the optimal pattern in a reasonable amount of time. However, we can see that $O(N^2)$+bb (the $O(N^2)$ algorithm with the branch and bound strategy Algorithm 2) is much faster for all input sizes. Although $O(N^2)$+simple bb (the $O(N^2)$ algorithm with the simple branch and bound strategy Algorithm 1) is fairly efficient as well, the extra work invested in the $O(N^2)+$bb algorithm is payed off by a $\sim 20\%$ reduction in the overall computation time.

To show that the algorithm also allows for the discovery of more complex patterns in a practical amount of time, we searched for the optimal don't care pattern on the same data set. The search took 765 minutes and 529 minutes, respectively, using simple bb and bb with a simple enumeration of don't care patterns, limiting the maximum length of the pattern to 15 and the number of don't cares characters in the pattern to 20% of its length.

5 Discussion

We presented efficient algorithmic solutions to the problem of discovering the optimal pattern in terms of a linear least squares fitting of the numeric attribute values associated with strings, and the matching function values. The efficiency of the algorithms are confirmed through computational experiments conducted on actual biological data.

In [20], the branch-and-bound enumerative search was applied in the *reverse* direction for finding the optimal degenerate pattern that discriminates between a positive string set and negative string set, where the matching function is an indicator function. In their search, the search tree is essentially traversed bottom-up. A bound on the score is computed in a similar way as for the original direction, and the traversal on the search can be pruned. This reverse direction is, however, difficult to achieve for the problem considered in this paper. This is because in the original direction, the bound is calculated from the numeric attribute values of strings which do not match the pattern ($\psi_p(s_i) = 0$), which is possible due to the condition $\psi_p(s_i) \geq 0$. In order to calculate a bound for the reverse direction, we would need to assume a maximum value c where $\psi_p(s_i) \leq c$, and we would calculate a bound for the residual sum of squares from the numeric attribute values of strings which give $\psi_p(s_i) = c$. However, the matching function used in this paper is not suitable, since the maximum value would change for each string.

The lower bound calculated in this paper underestimates the actual minimum value that can be achieved with the matching function and numeric attribute values. This is because we did not require that there exist a pattern whose matching function values would be equal to $\boldsymbol{x}^{\mathrm{opt}}$. Notice that when calculating the lower bound, $\boldsymbol{x}^{\mathrm{opt}}$ is obtained by considering points on the regression line. However, we know that the matching function will only take discrete integer values, and it may not be possible for some x_i^{opt} to lie on the regression line. Residuals for such points will not be zero, and would therefore increase the lower bound. Finding an *efficient* way to calculate a better lower bound for the discretized version of the problem is an interesting open problem.

Acknowledgements

Computation time was provided in part by the Super Computer System, Human Genome Center, Institute of Medical Science, University of Tokyo.

References

1. Brazma, A., Jonassen, I., Eidhammer, I., Gilbert, D.: Approaches to the automatic discovery of patterns in biosequences. J. Comput. Biol. **5** (1998) 279–305
2. Hirao, M., Hoshino, H., Shinohara, A., Takeda, M., Arikawa, S.: A practical algorithm to find the best subsequence patterns. Theoretical Computer Science **292** (2002) 465–479
3. Shinohara, A., Takeda, M., Arikawa, S., Hirao, M., Hoshino, H., Inenaga, S.: Finding best patterns practically. In: Progress in Discovery Science. Volume 2281 of LNAI., Springer-Verlag (2002) 307–317
4. Takeda, M., Inenaga, S., Bannai, H., Shinohara, A., Arikawa, S.: Discovering most classificatory patterns for very expressive pattern classes. In: 6th International Conference on Discovery Science (DS 2003). Volume 2843 of LNCS., Springer-Verlag (2003) 486–493

5. Hirao, M., Inenaga, S., Shinohara, A., Takeda, M., Arikawa, S.: A practical algorithm to find the best episode patterns. In: Proc. 4th International Conference on Discovery Science (DS2001). Volume 2226 of LNAI., Springer-Verlag (2001) 435–440
6. Inenaga, S., Bannai, H., Shinohara, A., Takeda, M., Arikawa, S.: Discovering best variable-length-don't-care patterns. In: Proceedings of the 5th International Conference on Discovery Science. Volume 2534 of LNAI., Springer-Verlag (2002) 86–97
7. Bussemaker, H.J., Li, H., Siggia, E.D.: Regulatory element detection using correlation with expression. Nature Genetics **27** (2001) 167–171
8. Bannai, H., Inenaga, S., Shinohara, A., Takeda, M., Miyano, S.: A string pattern regression algorithm and its application to pattern discovery in long introns. Genome Informatics **13** (2002) 3–11
9. Bannai, H., Inenaga, S., Shinohara, A., Takeda, M., Miyano, S.: Efficiently finding regulatory elements using correlation with gene expression. Journal of Bioinformatics and Computational Biology **2** (2004) 273–288
10. Zilberstein, C.B.Z., Eskin, E., Yakhini, Z.: Using expression data to discover RNA and DNA regulatory sequence motifs. In: The First Annual RECOMB Satellite Workshop on Regulatory Genomics. (2004)
11. Bannai, H., Hyyrö, H., Shinohara, A., Takeda, M., Nakai, K., Miyano, S.: An $O(N^2)$ algorithm for discovering optimal Boolean pattern pairs. IEEE/ACM Transactions on Computational Biology and Bioinformatics **1** (2004) 159–170 (special issue for selected papers of WABI 2004).
12. Hui, L.: Color set size problem with applications to string matching. In: Proceedings of the Third Annual Symposium on Combinatorial Pattern Matching (CPM 92). Volume 644 of LNCS., Springer-Verlag (1992) 230–243
13. Miyano, S., Shinohara, A., Shinohara, T.: Which classes of elementary formal systems are polynomial-time learnable? In: Proceedings of the 2nd Workshop on Algorithmic Learning Theory. (1991) 139–150
14. Miyano, S., Shinohara, A., Shinohara, T.: Polynomial-time learning of elementary formal systems. New Generation Computing **18** (2000) 217–242
15. Gusfield, D.: Algorithms on Strings, Trees, and Sequences. Cambridge University Press (1997)
16. Knuth, D.E., Morris, J.H., Pratt, V.R.: Fast pattern matching in strings. SIAM Journal on Computing **6** (1977) 323–350
17. Kasai, T., Lee, G., Arimura, H., Arikawa, S., Park, K.: Linear-time longest-common-prefix computation in suffix arrays and its applications. In: 12th Annual Symposium on Combinatorial Pattern Matching (CPM 2001). Volume 2089 of LNCS., Springer-Verlag (2001) 181–192
18. Kasai, T., Arimura, H., Arikawa, S.: Efficient substring traversal with suffix arrays. Technical Report 185, Department of Informatics, Kyushu University (2001)
19. Spellman, P.T., Sherlock, G., Zhang, M.Q., Iyer, V.R., Anders, K., Eisen, M.B., Brown, P.O., Botstein, D., Futcher, B.: Comprehensive identification of cell cycle-regulated genes of the yeast *Saccharomyces cerevisiae* by microarray hybridization. Mol. Biol. Cell **9** (1998) 3273–3297
20. Shinozaki, D., Akutsu, T., Maruyama, O.: Finding optimal degenerate patterns in DNA sequences. Bioinformatics **19** (2003) ii206–ii214