An $O(N^2)$ Algorithm for Discovering Optimal Boolean Pattern Pairs

Hideo Bannai, Heikki Hyyrö, Ayumi Shinohara, Masayuki Takeda, Kenta Nakai, and Satoru Miyano

Abstract—We consider the problem of finding the optimal *combination* of string patterns, which characterizes a given set of strings that have a numeric attribute value assigned to each string. Pattern combinations are scored based on the correlation between their occurrences in the strings and the numeric attribute values. The aim is to find the combination of patterns which is best with respect to an appropriate scoring function. We present an $O(N^2)$ time algorithm for finding the optimal *pair* of *substring patterns* combined with Boolean functions, where N is the total length of the sequences. The algorithm looks for all possible Boolean combinations of the patterns, e.g., patterns of the form $p \wedge \neg q$, which indicates that the pattern pair is considered to occur in a given string s, if p occurs in s, AND q does NOT occur in s. An efficient implementation using suffix arrays is presented, and we further show that the algorithm can be adapted to find the best $k\text{-pattern}$ Boolean combination in $O(N^k)$ time. The algorithm is applied to mRNA sequence data sets of moderate size combined with their turnover rates for the purpose of finding regulatory elements that *cooperate, complement, or compete with* each other in enhancing and/or silencing mRNA decay.

Index Terms—Pattern discovery, Boolean patterns, suffix tree, suffix array.

1 Introduction

Although recent genome sequencing projects have revealed the whole DNA sequence of several organisms, there is still much that is unknown concerning what and how the information is encoded in these blueprints of life. Pattern discovery from such biological sequences is thus an important topic in bioinformatics that has been studied heavily with numerous variations and applications (see [1] for a survey on earlier work). To extract meaning from biological sequences, the general goal of these methods is to find patterns which are conserved across a set of biologically related sequences. The existence of such sequence elements suggests that those elements are central to the functions and characteristics of the sequence set. Computational analyses which provide such candidates can be a very helpful guide for biologists in the task of experimentally confirming the actual sequence elements in play, as well as their functions.

Although finding the most significant sequence element conserved across multiple sequences has important applications, it is known that more than one sequence element will affect the biological characteristics of the sequences in many actual cases. There are several methods which address this observation, focusing on finding *composite* patterns. In [2], they develop a suffix tree-based approach for discovering *structured motifs*, which are two or more patterns separated by a certain distance, similar to text associative patterns [3]. MITRA [4] is another method that looks for composite patterns using *mismatch trees*. Bioprospector [5] applies the Gibbs sampling strategy to find gapped motifs. Multiple unordered motifs are considered in [6].

In this paper, we assume that we are given a set of sequences that have numeric attribute values associated with each sequence as input. We present a new formulation of composite pattern discovery where the problem is to find *pairs* of patterns combined with *any Boolean function*. The main contribution is an $O(N^2)$ algorithm (where N is the total length of the input strings) and implementation based on suffix arrays, for finding the *optimal* Boolean substring pattern pair with respect to some suitable scoring function. Note that the methods mentioned above for finding composite patterns can be viewed as being limited to finding pattern pairs which use only the $\wedge$ (AND) operation (with an extra distance constraint in the case of gapped motifs). In other words, the algorithms find combinations of two patterns p, q where both p AND q occur in each string. The use of any Boolean function permits the use of the $\neg$ (NOT) operation, allowing combinations such as $p \wedge \neg q$. This makes it possible to find not only sequence elements that *cooperate* with each other, but those with *competing* functions, i.e., not only the presence of one element, but the *absence* of the other is crucial for their functions. The pattern pairs discovered by our algorithm are optimal in that they are guaranteed to be the highest scoring pair of substring patterns with respect to a given scoring function and, also, a limit on the lengths of the patterns in the pair is not assumed. Our algorithm can be adjusted to handle several common problem formulations of pattern discovery, for example, pattern discovery from positive and negative

- H. Bannai, K. Nakai, and S. Miyano are with the Human Genome Center, Institute of Medical Science, The University of Tokyo, 4-6-1 Shirokanedai, Minato-ku, Tokyo 108-8639, Japan.
E-mail: {bannai, knakai, miyano}@ims.u-tokyo.ac.jp.
- H. Hyyrö is with PRESTO, Japan Science and Technology Agency (JST), Kawaguchi-shi, Saitama, Japan. E-mail: heikki.hyyro@gmail.com.
- A. Shinohara is with PRESTO, Japan Science and Technology Agency (JST) and the Department of Informatics, Graduate School of Information Science and Electrical Engineering, Kyushu University, 6-10-1 Hakozaki, Higashi-ku, Fukuoka 812-8581, Japan. E-mail: ayumi@i.kyushu-u.ac.jp.
- M. Takeda is with SORST, Japan Science and Technology Agency (JST) and the Department of Informatics, Graduate School of Information Science and Electrical Engineering, Kyushu University, 6-10-1 Hakozaki, Higashi-ku, Fukuoka 812-8581, Japan. E-mail: takeda@i.kyushu-u.ac.jp.

Manuscript received 3 Oct. 2004; revised 3 Dec. 2004; accepted 14 Dec. 2004. For information on obtaining reprints of this article, please send e-mail to: tcbb@computer.org, and reference IEEECS Log Number TCBBSI-0163-1004.

sequence sets [7], [8], [9], [10], as well as the discovery of patterns that *correlate* with a given numeric attribute (e.g., gene expression level) assigned to the sequences [11], [12], [13], [14], [15]. The significance of the algorithm in this paper lies in the fact that, since there are indeed $O(N^2)$ possible substring pattern combinations, the information needed to calculate the score for each pattern pair can be gathered, effectively, in constant time.

The algorithm is presented conceptually as using a generalized suffix tree [16], which is an indispensable data structure for efficient processing of substring information. Moreover, the algorithm using the suffix tree can be simulated very efficiently, with the same asymptotic complexity, using suffix arrays. We apply our algorithm to 3'UTR (untranslated region) of yeast and human mRNA, together with data obtained from microarray experiments which measure the decay rate of each mRNA [17], [18]. We were successful in obtaining several interesting pattern pairs where some correspond to known mRNA destabilizing elements.

A preliminary version of this paper appears in [19]. In this paper, we further present several generalizations of the problem and algorithm and show how to find the optimal k-pattern Boolean combination in $O(N^k)$ time, as well as the consideration of multiple string attributes as input.

2 Preliminaries

2.1 Notation

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 $length(w)$. The empty string is denoted by ε, that is, $length(\varepsilon) = 0$. The ith character of a string w is denoted by $w[i]$ for $1 \leq i \leq length(w)$ and the substring of a string w that begins at position i and ends at position j is denoted by $w[i:j]$ for $1 \leq i \leq j \leq length(w)$. For convenience, let $w[i:j] = \varepsilon$ for $j < i$. For any set S, let $|S|$ denote the cardinality of the set.

Let $\psi(p, s)$ be a Boolean matching function that has the value `true` if the *pattern* string p is a substring of the string s and `false` otherwise. We define the triplet $\langle F, p, q\rangle$ as a *Boolean pattern pair* (or simply *pattern pair*), which consists of two patterns, p and q, and a 2-ary Boolean function $F : \{\texttt{true}, \texttt{false}\} \times \{\texttt{true}, \texttt{false}\} \rightarrow \{\texttt{true}, \texttt{false}\}$. The matching function value $\psi(\langle F, p, q\rangle, s)$ is defined as $F(\psi(p, s), \psi(q, s))$. Table 1 lists all 16 possible Boolean functions of two Boolean variables, that is, all possible choices for F. We say that a pattern or Boolean pattern pair π *matches* string s if and only if $\psi(\pi, s) = \texttt{true}$. Note that the pattern ε matches any string.

For a given set of strings $S = \{s_1, \ldots, s_m\}$, let $M(\pi, S)$ denote the set of indices of strings in S that π matches, that is, $M(\pi, S) = \{i \mid \psi(\pi, s_i) = \texttt{true}\}$, and let its complement be denoted as $\overline{M}(\pi, S) = \{i \mid \psi(\pi, s_i) = \texttt{false}\}$. Now, suppose that, for each $s_i \in S$, we are given an associated numeric attribute value r_i. Let $R(\pi, S) = \sum_{i \in M(\pi,S)} r_i$ denote the sum of r_i over all s_i such that π matches. For brevity, we shall omit S where possible and let $M(\pi)$ and $R(\pi)$ be shorthand for $M(\pi, S)$ and $R(\pi, S)$, respectively. Note that $|M(\varepsilon)| = m$ and $R(\varepsilon) = \sum_{i=1}^{m} r_i$.

TABLE 1
Summary of Candidate Boolean Operations on Pattern Pair $\langle F, p, q\rangle$

	input				
$\psi(p,s)$	true	true	false	false	
$\psi(q,s)$	true	false	true	false	
	output $F(\psi(p,s),\psi(q,s))$				representation
F_0	false	false	false	false	false
F_1	false	false	false	true	$(\neg p) \wedge (\neg q)$
F_2	false	false	true	false	$(\neg p) \wedge q$
F_3	false	false	true	true	$(\neg p)$
F_4	false	true	false	false	$p \wedge (\neg q)$
F_5	false	true	false	true	$(\neg q)$
F_6	false	true	true	false	$(p \wedge (\neg q)) \vee ((\neg p) \wedge q)$
F_7	false	true	true	true	$(\neg p) \vee (\neg q)$
F_8	true	false	false	false	$p \wedge q$
F_9	true	false	false	true	$(p \wedge q) \vee ((\neg p) \wedge (\neg q))$
F_{10}	true	false	true	false	q
F_{11}	true	false	true	true	$(\neg p) \vee q$
F_{12}	true	true	false	false	p
F_{13}	true	true	false	true	$p \vee (\neg q)$
F_{14}	true	true	true	false	$p \vee q$
F_{15}	true	true	true	true	true

2.2 Problem Definition

In general, the problem of finding a good pattern from a given set of strings S refers to finding a pattern π that maximizes some suitable scoring function $score$ with respect to the strings in S. We concentrate on scoring functions whose values for a pattern π depend on values cumulated over the strings in S that match π. We also assume that the score value computation itself can be done in constant time if the required parameter values are known. More specifically, we concentrate on a $score$ that takes parameters of type $|M(\pi)|$ and $R(\pi)$. The specific choice of the scoring function highly depends on the particular application. A variety of problems fall into the category represented by the following problem definition:

Problem 1 (Optimal pair of substring patterns). *Given a set $S = \{s_1, \ldots, s_m\}$ of strings, where each string s_i is assigned a numeric attribute value r_i and a scoring function $score : \mathbf{R} \times \mathbf{R} \Rightarrow \mathbf{R}$, find the Boolean pattern pair $\pi \in \{\langle F, p, q\rangle \mid p, q \in \Sigma^*, F \in \{F_0, \ldots, F_{15}\}\}$ that maximizes $score(|M(\pi)|, R(\pi))$.*

Intuitively, the score for a given pattern π should be a measure of the difference between the two distributions of r_i, one corresponding to the set of strings that π matches and the other corresponding to the set that π does not match. A greater difference would mean that π is a better characterization, with respect to r_i, of the set of strings it matches. Many statistical measures for this purpose can be expressed as a function of $|M(\pi)|$ and $R(\pi)$. We give several examples of choices for a suitable $score$ and r_i below.

2.2.1 Positive/Negative Sequence Set Discrimination

We are given two disjoint sets of sequences S_1 and S_2, where sequences in S_1 (the positive set) are known to possess some biological function or characteristic, while the sequences in S_2 (the negative set) are known not to. The objective is to find pattern pairs which match more sequences in one set and less in the other.

We create an instance of the optimal pair of substring patterns problem as follows: Let $S = S_1 \cup S_2 = \{s_1, \ldots, s_m\}$ and let $r_i = 1$ if $s_i \in S_1$ and $r_i = 0$ if $s_i \in S_2$. Then, for each pattern pair π, the scoring function will receive $|M(\pi, S)|$ and $R(\pi, S) = |M(\pi, S_1)|$. Notice that $|M(\pi, S_2)| = |M(\pi, S)| - |M(\pi, S_1)|$. Common scoring functions that are used in this situation include the entropy information gain, the Gini index, and the chi-square statistic, which all are essentially functions of $|M(\pi, S_1)|$, $|M(\pi, S_2)|$, $|S_1|$, and $|S_2|$.

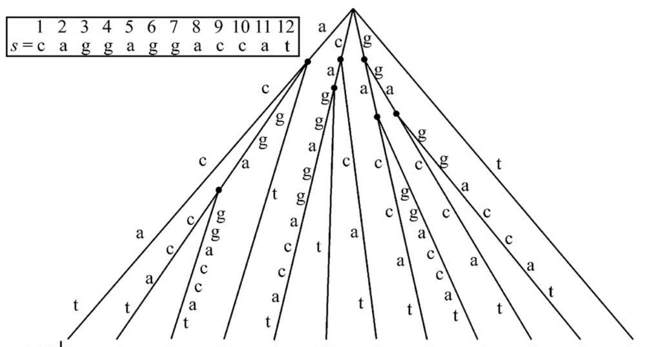
2.2.2 Correlated Patterns

We are given a set S of sequences, with a numeric attribute value r_i associated with each sequence $s_i \in S$, and the task is to find pattern pairs whose occurrences in the sequences *correlate* with their numeric attributes. For example, r_i could be the expression level ratio of a gene with upstream sequence s_i. The scoring function used in [12], [14] is the interclass variance, which can be maximized by maximizing the scoring function $score(x, y) = y^2/x + (y - \sum_{i=1}^{m} r_i)^2/(m - x)$, where $x = |M(\pi)|$ and $y = R(\pi)$. We will later describe how to construct a nonparametric scoring function based on the normal approximation of the Wilcoxon rank sum test, which can also be used in our framework.

2.3 Basic Data Structures

A *suffix tree* [16] 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 in 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 m strings $S = \{s_1, \ldots, s_m\}$ is basically a suffix tree for the string $s_1\$_1 \cdots s_m\$_m$, where each $\$_i$ $(1 \leq i \leq m)$ 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. It is well-known that suffix trees (and generalized suffix trees) can be represented in linear space and constructed in linear time [16] with respect to the length of the string (total length of the strings for GST).

A *suffix array* [20] A_s for a given string s of length n is a permutation of the integers $1, \ldots, n$ representing the lexicographic ordering of the suffixes of s. The value $A_s[i] = j$ in the array indicates that $s[j : n]$ is the ith suffix in the lexicographic ordering. The *lcp array* for a given string s is an array of integers representing the longest common prefix lengths of adjacent suffixes in the suffix array. We define $lcp_s[1] = 0$, $lcp_s[i] = \max\{k \mid s[A_s[i-1] : A_s[i-1] + k - 1] = s[A_s[i] : A_s[i] + k - 1]\}$ for $2 \leq i \leq n$, and $lcp_s[i] = -1$ otherwise. Recently, three methods for constructing the suffix array directly from a string in linear time have been developed [21], [22], [23]. The lcp array can be constructed from the suffix array also in linear time [24]. It has been shown that several algorithms (and potentially many more) which utilize the suffix tree can be implemented very efficiently using the suffix array together with its lcp array [24], [25] (the combination termed, in [25], the *enhanced suffix array*). This paper presents yet another example for efficient implementation of an algorithm based conceptually on suffix trees, but uses the suffix and lcp arrays.



$A_s[i]$	8	5	2	11	1	10	9	7	4	6	3	12
$lcp_s[i]$	0	1	4	1	0	2	1	0	2	1	3	0
i	1	2	3	4	5	6	7	8	9	10	11	12

Fig. 1. A suffix tree, suffix array A_s, and lcp array lcp_s for string $s = \texttt{caggaggaccat}$. Notice that the paths of the suffix tree from the root to the leaves (i.e., suffixes) are sorted in lexicographic order from left to right, each leaf corresponding to a position in the suffix array. The integer in the suffix array represents the position in the string from which the corresponding suffix starts. The lcp array represents the length of the longest path that consecutive suffixes in the suffix array share.

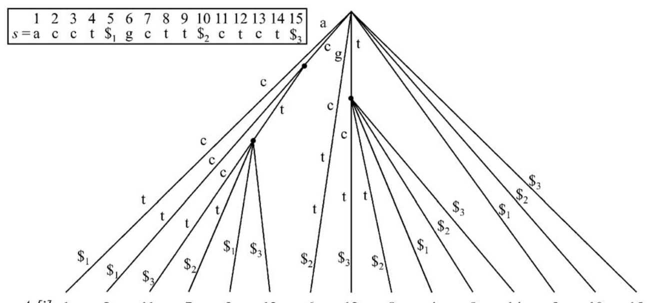
The *lowest common ancestor* $lca(x, y)$ of any two nodes x and y in a tree is the deepest node which is common to the paths from the root to each of the nodes. The tree can be preprocessed in linear time to answer the lowest common ancestor (*lca-query*) for any given pair of nodes in constant time [26]. In terms of the suffix array, the lca-query is almost equivalent to a *range minimum query* (rm-query) on the lcp array. Given a pair of positions i and j, an rm-query $rmq(i, j)$ on the lcp array returns the position of the minimum element in the subarray $lcp[i : j]$. The lcp array can also be preprocessed in linear time to answer the rm-query in constant time [26], [27].

Figs. 1 and 2 show examples of a suffix tree and generalized suffix tree, as well as their corresponding suffix arrays and lcp arrays.

The linear time bounds mentioned above for the construction of suffix trees and arrays, as well as the preprocessing for lca- and rm-queries, are actually not required for the $O(N^2)$ overall time bound for finding optimal pattern pairs. This is because the results of all queries can be calculated naively in $O(N^2)$ time once and their results stored for reuse. However, they are very important for an efficient implementation of our algorithm.

3 Algorithm

Now, we present algorithms to solve the optimal pair of substring patterns problem, given the set of strings $S = \{s_1, \ldots, s_m\}$, an associated attribute r_i for each string s_i, and a scoring function $score$. Also, let $N = \sum_{i=1}^{m} length(s_i)$.



$A_s[i]$	1	2	11	7	3	13	6	12	8	4	9	14	5	10	15
$lcp_s[i]$	0	0	1	2	2	2	0	0	1	1	1	1	0	0	0
id	1	1	3	2	1	3	2	3	2	1	2	3	1	2	3
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Fig. 2. A generalized suffix tree and its corresponding suffix array for the strings $\{\texttt{acct}, \texttt{gctt}, \texttt{ctct}\}$.

We first show that a naive algorithm requires $O(N^3)$ time and then describe the $O(N^2)$ algorithm. The algorithms calculate scores for all possible combinations of pattern pairs, from which finding the optimal pair is a trivial task.

3.1 An $O(N^3)$ Algorithm

We know that we only need to consider $O(N)$ candidates for a single pattern since the candidates can be confined to patterns of form $l(v)$, where v is a node in the generalized suffix tree over the set S. This is because, for any pattern corresponding to a path that ends in the middle of an edge of the suffix tree, the pattern which corresponds to the path extended to the next node will match the same set of strings and, hence, the score would be the same. Therefore, there are $O(N^2)$ possible candidate pattern pairs for which we must calculate the scoring function value. For a given pattern pair candidate $\pi = \langle F, l(v_1), l(v_2)\rangle$, where v_1, v_2 are nodes of the GST, the values $|M(\pi)|$ and $R(\pi)$ can be computed in $O(N)$ time by using any of the linear time string matching algorithms. Then, each corresponding scoring function value can be computed in constant time. Therefore, the total time required is $O(N^3)$, using $O(N)$ space for the generalized suffix tree.

The time complexity can be further improved to $O(mN^2)$ as follows: For each pattern candidate p, we store the matching function values $\psi(p, s_1), \dots, \psi(p, s_m)$ as an array of length m. This can be computed using a linear time string matching algorithm, taking $O(N)$ time for each pattern candidate, for a total of $O(N^2)$ time to calculate all $O(N)$ arrays. With this precalculation, the score for a given pattern pair $\pi = \langle F, p, q\rangle$ can be calculated in $O(m)$ time by a single loop over $i = 1, \dots, m$ to accumulate values according to $F(\psi(p, s_i), \psi(q, s_i))$ to obtain $|M(\pi)|$ and $R(\pi)$. The total time would then be $O(mN^2)$ time to calculate scores for all pattern pairs, which could be reasonable for small m, but would still be prohibiting otherwise. The space complexity is also increased to $O(mN)$ for storing the arrays of length m.

3.2 An $O(N^2)$ Algorithm

Our algorithm is derived from the technique for solving the *color set size problem* [28], which calculates the values $|M(l(v))|$ in $O(N)$ time for all nodes v of a GST over the string set S. Let us first describe a slight generalization of this algorithm, described in [14].

Lemma 1. *Given a set of strings $S = \{s_1, \dots, s_m\}$, corresponding numeric attributes r_i for each s_i, and a GST of S, $|M(l(v))|$ and $R(l(v))$ can be computed for all nodes v of the GST in, total of $O(N)$ time and space.*

Proof. The following algorithm computes the values $R(l(v))$ for all nodes v in the GST. Note that if we give each attribute r_i the value 1, then $R(l(v)) = |M(l(v))|$. Thus, we do not need to consider separately how to compute $|M(l(v))|$.

First, we introduce some auxiliary notation. Let $LF(v)$ denote the set of all leaf nodes in the subtree rooted by the node v and let $c_i(v)$ denote the number of leaves in $LF(v)$ that have the label id_i. Let us also define the sum of leaf attributes for a node v as $\sum_{LF(v)} r_i$. Since $LF(v)$ corresponds to all occurrences of $l(v)$ in the string set S, we have that

$$\sum_{LF(v)} r_i = \sum_{I \in M(l(v))} (c_i(v) \cdot r_i). \tag{1}$$

For any node v in the GST over the string set S, the matching value $\psi(l(v), s_i)$ is $\texttt{true}$ for at least one string s_i. Thus, the equality

$$R(l(v)) = \sum_{I \in M(l(v))} r_i = \sum_{LF(v)} r_i - \sum_{I \in M(l(v))} ((c_i(v) - 1) \cdot r_i) \tag{2}$$

holds. Let us define the preceding subtracted sum to be a *correction factor*, which we denote by

$$corr(l(v), S) = \sum_{i \in M(l(v))} ((c_i(v) - 1) \cdot r_i). \tag{3}$$

Since the recurrence

$$\sum_{LF(v)} r_i = \sum_{v'} \left(\sum_{LF(v')} r_i \mid v' \text{ is a child node of } v \right) \tag{4}$$

clearly holds, the values $\sum_{LF(v)} r_i$ can be easily calculated for all v during a linear time bottom-up (postorder) traversal of the GST.

The next step is to remove the redundancies, represented by the values $corr(l(v), S)$, from the values $\sum_{LF(v)} r_i$. Let $I(id_i)$ be the list of all leaves with the label id_i in the order they appear in a postorder traversal of the tree. Clearly, the lists $I(id_i)$ can be constructed in linear time for all labels id_i. We note the following four simple but useful properties:

1. The leaves in $LF(v)$ with the label id_i form a continuous interval of length $c_i(v)$ in the list $I(id_i)$.
2. If $c_i(v) > 0$, a length-$c_i(v)$ interval in $I(id_i)$ contains $c_i(v) - 1$ adjacent (overlapping) leaf pairs.
3. If $x, y \in LF(v)$, the node $lca(x, y)$ belongs to the subtree rooted by v.
4. For any $s_i \in S$, $\psi(l(v), s_i) = \texttt{true}$, that is, $i \in M(l(v))$ if and only if there is a leaf $x \in LF(v)$ with the label id_i.

Assume that each node v has a correction value that has been initialized to 0. Consider now what happens if we go through all adjacent leaf pairs x, y in the list $I(id_i)$ and add, for each pair, the value r_i into the correction value of the

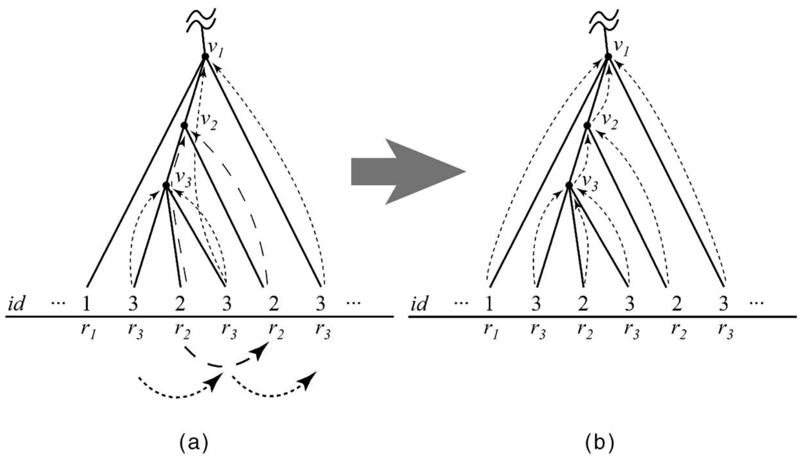


Fig. 3. Illustration of linear time algorithm for calculating the the sum of weights of distinct ids in the subtree of each node. First, correction factors are set at the lca of consecutive leaves of the same id. This sets the correction values at internal nodes v_1, v_2, v_3 to r_3, r_2, and r_3, respectively (a). Then, with the bottom-up (postorder) traversal (b), the sums accumulated at v_3, v_2, v_1 become $r_3 + r_2 + r_3 - r_3 = r_2 + r_3 = R(l(v_3))$, $R(l(v_3)) + r_2 - r_2 = r_2 + r_3 = R(l(v_2))$, and $r_1 + R(l(v_2)) + r_3 - r_3 = r_1 + r_2 + r_3 = R(l(v_1))$, respectively, as desired. (a) Store correction factors at the lca of adjacent leaves of same id. (b) Propagate leaf weights and correction factors upward with a bottom-up (postorder) traversal.

node $lca(x, y)$. It follows from Properties 1-3 that now, for each node v in the tree, the sum of the correction values in the nodes of the subtree rooted by v equals $(c_i(v) - 1) \cdot r_i$. Moreover, if we repeat the process for each of the lists $I(id_i)$, then, due to Property 4, the preceding total sum of the correction values in the subtree rooted by v becomes $\sum_{i \in M(l(v))}((c_i(v) - 1) \cdot r_i) = corr(l(v), S)$. Hence, at this point, a single linear time bottom-up (postorder) traversal of the tree enables us to cumulate the correction values $corr(l(v), S)$ from the subtrees into each node v and, at the same time, we may record the final values $R(l(v))$. This procedure is illustrated in Fig. 3.

The preceding process involves a constant number of linear time traversals of the tree, as well as a linear number of lca-queries. Since each lca-query can be done in constant time after a linear time preprocessing, the total time for computing the values $R(l(v))$ for all nodes v is linear.

The linear time algorithm is shown as pseudocode in Fig. 4. □

The above-described algorithm permits us to compute the values $R(l(v))$ and $|M(l(v))|$ in linear time, which, in turn, leads into a linear time solution for the problem of finding the best pattern when the pattern is a *single* substring: The scoring function can now be computed for each possible pattern candidate $l(v)$. The case of a Boolean pattern pair will be solved in a similar manner, that is, we will concentrate on how to compute the values $R(\pi)$ (and $|M(\pi)|$) for all possible $O(N^2)$ pattern pair candidates, where $\pi = \langle F, l(v_1), l(v_2)\rangle$ and v_1, v_2 are any two nodes in the GST over S. If we manage to do this in $O(N^2)$ time, then the whole problem will be solved in $O(N^2)$ under the assumption that the scoring function can be computed in constant time for each candidate.

Naive use of the information gathered by the single substring pattern algorithm is not sufficient for solving the problem for *pairs* of patterns in $O(N^2)$ time. This is because, in order to compute the needed values $|M(\langle F, l(v_1), l(v_2)\rangle)|$ and $R(\langle F, l(v_1), l(v_2)\rangle)$ from $|M(l(v_1))|, |M(l(v_2))|$ and $R(l(v_1))$, $R(l(v_2))$, we must somehow conduct an intrinsic set operation between the string subsets that match or do not match $l(v_1)$ and $l(v_2)$. However, an $O(N^2)$ algorithm for pattern pairs is fairly simple to derive from the linear time algorithm for the single pattern.

Theorem 1. *The optimal pair of substring patterns problem can be solved in $O(N^2)$ time and $O(N)$ space for any scoring function score provided that it can be calculated in constant time given its inputs.*

Proof. We go over the $O(N)$ choices for the first pattern, $l(v_1)$. For each such fixed $l(v_1)$, we use a modified version of the linear time algorithm shown above in order to process the $O(N)$ choices for the second pattern $l(v_2)$ in $O(N)$ time. More precisely, given a fixed $l(v_1)$, we additionally label each string $s_i \in S$ and the corresponding leaves in the GST with the Boolean value $\psi(l(v_1), s_i)$. This can be done in $O(N)$ time using any linear time string matching algorithm. Now, the trick is to cumulate the sums and correction factors *separately* for different values of the additional label. The end result is that we will have values

$$\begin{aligned}&\sum_{i \in M(l(v_2))} (r_i \mid \psi(l(v_1), s_i) = \texttt{true})\\ &= \sum_i (r_i \mid \psi(l(v_1), s_i) = \texttt{true}, \psi(l(v_2), s_i) = \texttt{true})\\ &= R(\langle F_8, l(v_1), l(v_2)\rangle)\end{aligned}$$

and

$$\begin{aligned}&\sum_{i \in M(l(v_2))} (r_i \mid \psi(l(v_1), s_i) = \texttt{false})\\ &= \sum_i (r_i \mid \psi(l(v_1), s_i) = \texttt{false}, \psi(l(v_2), s_i) = \texttt{true})\\ &= R(\langle F_2, l(v_1), l(v_2)\rangle),\end{aligned}$$

which are decompositions of $\sum_{i \in M(l(v_2))} r_i = R(l(v_2))$ according to $\psi(l(v_1), s_i)$ for all nodes v in linear time. We note that

```
1  Build $GST(S)$ for $S = \{s_1, \ldots, s_m\}$;
2  Preprocess $GST(S)$ for $lca$-query;
3  foreach $i = 1, \ldots, m$ do: $I(id_i) = emptylist$; end
4  foreach node $v \in GST(S)$ in postorder do:
5     if $v$ is a leaf with label $id_i$ then append $v$ to $I(id_i)$; endif
6  end
7  foreach node $v \in GST(S)$ do:
8     if $v$ is an internal node then $R(l(v)) = 0$;
9     else /* $v$ is a leaf with label $id_i$ */ $R(l(v)) = r_i$; endif
10 end
11 foreach $i = 1, \ldots, m$ do:
12    foreach consecutive leaf pair $x, y$ in $I(id_i)$ do:
13       $R(lca(x,y))$ = $R(lca(x,y)) - r_i$;
14    end
15 end
16 foreach node $v \in GST(S)$ in postorder do:
17    $R(l(v)) = R(l(v)) + \sum_{v' \in children(v)} R(l(v'))$;
18    Calculate and report score for node $v$;
19 end
```

Fig. 4. Summary of the algorithm for solving the general version of the *color set size problem*, which calculates $R(l(v))$ for all nodes v. Note that $|M(l(v))|$ can be calculated for all nodes v by setting $r_i = 1$ for all $i = 1, \ldots, m$ and is not shown. In line 17, $children(v)$ represents the set of child nodes of node v. The score for each node v is calculated from $R(l(v))$ and $|M(l(v))|$ and reported at line 18.

$$\begin{aligned}
&\sum_{i \in \overline{M}(l(v_2))} (r_i \mid \psi(l(v_1), s_i) = \texttt{true}) \\
&= \sum_i (r_i \mid \psi(l(v_1), s_i) = \texttt{true}, \psi(l(v_2), s_i) = \texttt{false}) \\
&= R(\langle F_4, l(v_1), l(v_2) \rangle) \\
&= R(l(v_1)) - R(\langle F_8, l(v_1), l(v_2) \rangle)
\end{aligned}$$

and

$$\begin{aligned}
&\sum_{i \in \overline{M}(l(v_2))} (r_i \mid \psi(l(v_1), s_i) = \texttt{false}) \\
&= \sum_i (r_i \mid \psi(l(v_1), s_i) = \texttt{false}, \psi(l(v_2), s_i) = \texttt{false}) \\
&= R(\langle F_1, l(v_1), l(v_2) \rangle) \\
&= R(\epsilon) - R(l(v_1)) - R(\langle F_2, l(v_1), l(v_2) \rangle),
\end{aligned}$$

where the values $R(\varepsilon)$ and $R(l(v_1))$ can be easily computed in linear time. Thus, *all* cumulative values of the form $\sum_i (r_i \mid \psi(l(v_1), s_i) = \mathtt{b}_1, \psi(l(v_2), s_i) = \mathtt{b}_2)$, where $\mathtt{b}_1, \mathtt{b}_2 \in \{\texttt{true}, \texttt{false}\}$, can be computed in linear time. From these four values, it is straightforward to compute the values

$$\begin{aligned}
R(\langle F, l(v_1), l(v_2) \rangle) &= \sum_{i \in M(\langle F, l(v_1), l(v_2) \rangle)} r_i \\
&= \sum_i (r_i \mid F(\psi(l(v_1), s_i), \psi(l(v_2), s_i)) = \texttt{true}),
\end{aligned}$$

as well as the corresponding scoring function values, for all other $F \in \{F_0, \ldots, F_{15}\}$ in linear time. Thus, given a fixed $l(v_1)$, we can compute the scores for all pattern pair candidates of form $\langle F, l(v_1), l(v_2) \rangle$ in $O(N)$ time. Since there are only $O(N)$ candidates for $l(v_1)$, we have an $O(N^2)$ algorithm for evaluating all possible pattern pair candidates for any given $F \in \{F_0, \ldots, F_{15}\}$.

Since the $O(N)$ time calculations for each fixed $l(v_1)$ are independent of each other, the generalized suffix tree can be reused. Therefore, the space complexity of the algorithm is $O(N)$. The outline of the algorithm is shown as pseudocode in Fig. 5. □

The algorithm can be adapted to the general case of combining $k > 2$ patterns. We define the $(k+1)$-tuple $\langle F, p_1, \ldots, p_k \rangle$ as a k-pattern *Boolean combination* where F is a k-ary Boolean function and $p_1, \ldots, p_k$ are substring patterns. We say p_i is the ith *component* of the k-pattern Boolean combination. The matching function for a k-pattern Boolean combination $\pi = \langle F, p_1, \ldots, p_k \rangle$ is defined naturally as $\psi(\pi, s) = F(\psi(p_1, s), \ldots, \psi(p_k, s))$.

Corollary 1. *For a given k-ary ($k > 2$) Boolean function F, the optimal k-pattern combination $\pi = \langle F, p_1, \ldots, p_k \rangle$ can be found in $O(N^k)$ time and $O(N + mk)$ space for any scoring function score provided that it can be calculated in constant time given its inputs.*

Proof. For a given k-ary Boolean function F, we can decompose F into a sequence of 2-ary Boolean functions $G_1, \ldots, G_{k-1}$ such that

$$F(x_1, \ldots, x_k) \equiv G_{k-1}(G_{k-2}(\cdots G_1(x_1, x_2) \cdots), x_k)$$

```
1  Build $GST(S)$ for $S = \{s_1, \dots, s_m\}$;
2  $R(\varepsilon) = \sum_{i=1}^{m} r_i$;
3  foreach node $v_1 \in GST(S)$ do:
4      $R(l(v_1)) = 0$;
5      foreach $i = 1, \dots, m$ do:
6          Calculate $\psi(l(v_1), s_i)$;
7          if $\psi(l(v_1), s_i)$ == true then $R(l(v_1)) = R(l(v_1)) + r_i$; endif
8      end
9      foreach node $v_2 \in GST(S)$ do: /* Use variation of css algorithm */
10         Obtain $R(l(v_2))$ separately as $R(\langle F_8, l(v_1), l(v_2)\rangle)$ and $R(\langle F_2, l(v_1), l(v_2)\rangle)$
11         Calculate $R(\langle F, l(v_1), l(v_2)\rangle)$ and scores for all $F \in \{F_0, \dots, F_{15}\}$;
12     end
13 end
```

Fig. 5. Summary of the algorithm for solving the general version of the *color set size problem* for Boolean substring pattern pairs. The loop in lines 9 to 12 uses a variation of the algorithm in Fig. 4, where the sums for r_i are maintained separately for sequences with $\psi(l(v_1), s_i) = \texttt{true}$ and $\psi(l(v_1), s_i) = \texttt{false}$. In line 11, the value $R(\langle F, l(v_1), l(v_2)\rangle)$ can be calculated from $R(\varepsilon)$, $R(l(v_1))$, $R(\langle F_8, l(v_1), l(v_2)\rangle)$, and $R(\langle F_2, l(v_1), l(v_2)\rangle)$.

for all inputs $x_1, \dots, x_k \in \{\texttt{true}, \texttt{false}\}$. For a fixed node v_1 for the first pattern component, we label each string s_i and the corresponding leaves of the GST with the label $\psi(l(v_1), s_i)$, which can be done in $O(N)$ time. For $j = 2, \dots, k-2$, we repeat this process, this time labeling the strings and leaves with G_j, using the previous label and the Boolean value $\psi(l(v_j), s_i)$ as input. This can also be done in $O(N)$ time for each j. For the kth pattern component, the linear time algorithm for solving the color set size can be used with function G_{k-1} and the labels of the suffix tree obtained in the previous steps. Since there are at most $O(N)$ candidates for any given component of the pattern combination, the total time for considering all possible pattern combinations is therefore the sum of the nested loops:

$$O(N) \cdot [O(N) + O(N) \cdot [O(N) + O(N) \cdot [O(N) + \cdots]]] = O\left(\sum_{i=2}^{k} N^i\right) = O(N^k). \tag{5}$$

Since the suffix tree can be reused, the space complexity is $O(N)$ plus an extra $O(m)$ in each loop to remember the labels of each string. Note that choosing the optimal k-ary function for F would take an additional factor of $O(2^{2^k})$, the number of such functions. □

4 Implementation Using Suffix Arrays

The algorithm on the suffix tree can be simulated efficiently by a suffix array. We modify the algorithm of [24], [29] that simulates a bottom-up (postorder) traversal of a suffix tree using a suffix array. A subtlety in the modification lies in calculating the lca, as well as determining where to store the correction factor, which should be set at the lca since the simulation via suffix arrays does not explicitly create the internal nodes of the suffix tree. Notice that, since each suffix of the string corresponds to a leaf in the suffix tree, each leaf in the suffix tree corresponds to a position in the suffix array. Let us denote this position for a leaf x as $pos(x)$. The lowest common ancestor query between two leaves is conceptually equivalent to a range minimum query on the lcp array: For a given pair of leaves x, y such that $pos(x) < pos(y)$, we have that $length(l(lca(x, y))) = lcp[rmq(pos(x) + 1, pos(y))]$.

For storing the correction factors, we construct another array CF of the same length as the suffix array, representing internal nodes of the suffix tree. The correction factors $CF[...]$ are first initialized to 0 and, when setting the correction factor for two leaves x, y such that $pos(x) < pos(y)$, the correction value is added into $CF[rmq(pos(x) + 1, pos(y))]$.

Fig. 6 shows pseudocode for the modified version of the `Substring_Statistics` algorithm of [24], which originally reports $\sum_{LF(v)} r_i$ instead of $R(l(v))$ for each node v of the generalized suffix tree. The difference is in lines 14 and 17, where the correction factor $CF[i]$ is subtracted from the sums. In the ith step, the correction factor $CF[i]$ is subtracted from the (potentially) new node $lca(pos^{-1}(i-1), pos^{-1}(i))$, where

$$length(l(lca(pos^{-1}(i-1), pos^{-1}(i)))) = lcp[i]. \tag{6}$$

If $CF[i]$ is not zero, this means that there existed leaves x, y where $pos(x) \le i - 1 < i \le pos(y)$ such that $rmq(pos(x) + 1, pos(y)) = i$, and

$$length(l(lca(x, y))) = lcp[i]. \tag{7}$$

From (6) and (7), we have that $lca(x, y) = lca(pos^{-1}(i-1), pos^{-1}(i))$ and we can see that the correction factor is subtracted from the correct node.

5 Computational Experiments

5.1 Running Times

The algorithm was implemented using the C++ language. All results reported in this paper were computed on a Sun Fire 15K (UltraSPARC III Cu 1.2GHz x 96 CPUs). Table 2 shows the comparison of running times between the naive

```
Set correction factors to $CF[\ldots]$;
Let $Stack = \{(0,-1,0)\}$ be the stack;
foreach $i = 1,\ldots,N+1$ do:
    $(L_i, H_i, R_i) := (i, lcp[i], 0)$;  /* lca of leaf i-1 and i: (potential) new node */
    $(L, H, R) := top(Stack)$;    /* copy top element */
    while $(H > H_i)$ do:     /* report all nodes deeper than new node */
        $R_i := R + R_i$;              /* propagate sums from child node */
        report $(L, H, R_i)$;          /* report that $R(l(s[A[L] : A[L] + H - 1])) = R_i$ */
        $pop(Stack)$;                  /* remove top element */
        $(L, H, R) := top(Stack)$;     /* copy top element */
    end
    if ($i$ == $N + 1$) then exit loop endif;
    if $(H < H_i)$ then       /* put new node in stack */
        $push((L_i, H_i, R_i - CF[i]), Stack)$;
    else         /* $H = H_i$ : same node - update node on stack top */
        $(L, H, R) := top(Stack)$; $pop(Stack)$;    /* remove top element */
        $push((L, H, R + R_i - CF[i]), Stack)$;    /* reinsert with new values */
    endif
    $push((i, N - A[i] + 1, r_{id[i]}), Stack)$; /* insert new leaf in stack */
end
```

Fig. 6. Core of the algorithm for solving the general version of the *color set size problem* using a suffix array. We assume the correction factors are stored in the array CF. The algorithm simulates a postorder traversal on the suffix tree using the suffix array and corresponds to the loop in lines 16-19 of Fig. 4. A node v in the suffix tree is represented by a three-tuple (L, H, R), where L denotes the position in the suffix array for a leaf in $LF(v)$, H denotes the length of the path from the root to v, and R denotes $R(l(v))$.

$O(mN^2)$ algorithm and our $O(N^2)$ algorithm for the data set presented in Section 5.2.1. Our $O(N^2)$ algorithm is clearly faster.

Our algorithm is also highly parallelizable, which is shown by the running times and speed-up when varying the number of processors in the parallel implementation of our algorithm (Fig. 7). POSIX threads were used to execute parallel computations. Since the suffix tree (suffix array) traversal takes roughly the same time for each fixed first candidate pattern, the work load is simply divided into equal sized sets of first candidate patterns which each thread will compute and the results of each thread are combined later.

5.2 Finding Sequence Elements which Determine mRNA Degradation Rates

The degradation of mRNA, in addition to transcription, is one of several important mechanisms which control the expression level of a gene (see [30] for survey). The half lives of mRNA are very diverse: Some mRNAs can degrade 100 times faster than others, which allows their expression level to be adjusted more quickly. The degradation of mRNA is controlled by many factors, for example, it is known that some proteins bind to the UTR of the mRNA to promote its decay, while others inhibit it. Recently, the comprehensive decay rates of many genes have been measured using microarray technology [17], [18]. We consider the problem of finding substring pattern pairs related to the rate of mRNA decay to find possible binding sites of the proteins in order to further understand this complex mechanism.

TABLE 2
Approximate Running Times of Naive $O(mN^2)$ Algorithm and Our $O(N^2)$ Algorithm

algorithm	time
$O(mN^2)$	1885 min
$O(N^2)$	57 min

Measured with data in Section 5.2.1 ($N = 77200$, $m = 772$).

In the experiments presented, we limit the search to Boolean functions $F \in \{F_1, F_2, F_4, F_7, F_8, F_{11}, F_{13}, F_{14}\}$ because: F_0 and F_{15} are constant functions and clearly do not have discriminative power, F_3, F_5, F_{10}, F_{12} essentially ignore the matching results of one of the patterns in the pair and are not of interest to us in this paper. We also did not consider F_6, F_9, since it may be difficult to interpret their meaning biologically. Furthermore, for function pair F_i, F_j, where $F_i(\psi(p,s), \psi(q,s)) \equiv F_j(\psi(q,s), \psi(p,s))$ (F_2 and F_4, F_{11} and F_{13}), only one function per pair needs to be considered since all $O(N)$ candidates for p and q are considered. Also, for function pair F_i, F_j, where $F_i(\psi(p,s), \psi(q,s)) \equiv \neg F_j(\psi(p,s), \psi(q,s))$ (F_1 and F_{14}, F_2 and F_{13}, F_4 and F_{11}, F_7 and F_8), only one function per pair needs to be considered if $score$ is symmetric with respect to $|S|$ and $\sum_{i=1}^{m} r_i$, that is, if $score(|M(\pi)|, R(\pi)) = score(|S| - |M(\pi)|, (\sum_{i=1}^{m} r_i) - R(\pi))$.

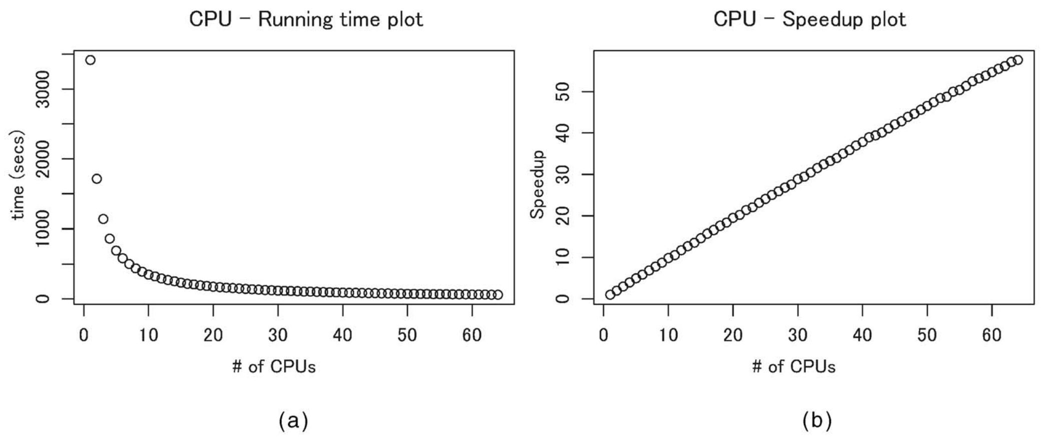


Fig. 7. The (a) running time and (b) speed-up plots of our algorithm using various numbers of CPUs for the data in Section 5.2.1. The algorithm can be highly parallelized and speedup is almost linear in the number of processors used.

5.2.1 *Positive/Negative Set Discrimination of Yeast Sequences*

For our first experiment, we used the two sets of predicted 3'UTR processing site sequences provided in [31], which are constructed based on the microarray experiments in [17] that measure the degradation rate of yeast mRNA. One set S_f consists of 393 sequences which have a fast degradation rate ($t_{1/2} < 10$ minutes), while the other set S_s consists of 379 predicted 3'UTR processing site sequences which have a slow degradation rate ($t_{1/2} > 50$ minutes). Each sequence is 100 nt long and the total length of the sequences is $77,200$ nt. The traversal on the suffix array on this data set shows that there are $46,554$ candidates for a single pattern (i.e., the number of internal nodes in the suffix tree. Patterns corresponding to leaf nodes were ignored since they are not "commonly occurring" patterns), meaning that there are $46,554^2 = 2,167,274,916$ possible pattern pairs. For the scoring function, we used the standard chi-squared statistic, calculated by

$$(|S_f| + |S_s|)\frac{(\mathtt{tp} * \mathtt{tn} - \mathtt{fp} * \mathtt{fn})^2}{(\mathtt{tp}+\mathtt{fn})(\mathtt{tp}+\mathtt{fp})(\mathtt{tn}+\mathtt{fp})(\mathtt{tn}+\mathtt{fn})}, \tag{8}$$

where $\mathtt{tp} = |M(\pi, S_f)|$, $\mathtt{fp} = |S_f| - \mathtt{tp}$, $\mathtt{tn} = |S_s| - \mathtt{fn}$, and $\mathtt{fn} = |M(\pi, S_s)|$. All four values may be calculated by setting r_i as shown in Section 2.2.1.

The top five scoring pattern pairs found are shown in Table 3. Several interesting patterns can be found in these pattern pairs. For all the patterns in the pairs that match more in the faster decaying set, the substring UGUA is contained. This sequence is actually known as a core consensus for the binding site of the PUF protein family that plays important roles in mRNA regulation [32] and has also been found in the previous analysis [31] to be significantly overrepresented in the fast degrading set.

On the other hand, patterns which are combined with $\neg$ can be considered as sequence elements which *compete* with UGUA and interfere with mRNA decay. The patterns AUCC and GUUG were in fact found to be substrings of a less studied mRNA *stabilizer* element, experimentally shown to be within a region of 65nt in the TEF1/2 transcripts [33]. We cannot say directly that the two substrings represent components of this stabilizer element since it was reported that this stabilizer element should be in the translated region in order to function. However, the mechanisms of stabilizers are not yet well understood and further investigation may uncover relationships between these sequences.

5.2.2 *Finding Correlated Patterns from Human Sequences*

For our second experiment, we used the decay rate measurements of the human hepatocellular carcinoma cell line HepG2 made available as Supplementary Table 9 of [18]. 3'UTR sequences for each mRNA was retrieved using the ENSMART [34] interface. We were able to obtain $2,306$ pairs of 3'UTR sequences and their decay rates, with the average length of the sequences being 925.54 nt, and the total length was $2,134,294$ nt.

Since the distribution of the turnover rates seemed to have a heavier tail than the normal distribution, we used a nonparametric scoring function that fits into our $O(N^2)$ total time bound: the normal approximation of the Wilcoxon rank sum test statistics. The set of sequences S is first sorted in increasing order according to its decay rate and each sequence s_i is assigned its rank for r_i. For a pattern pair π, the rank sum statistic $R(\pi) = \sum_{i \in M(\pi)} r_i$ approximately depends on the normal distribution when the sample size is large. Therefore, we use the z-score defined by:

$$z(x, y) = \frac{(y - x(|S|+1)/2)}{\sqrt{x(|S|-x)(|S|+1)/12}}, \tag{9}$$

TABLE 3
Top Five Scoring Pattern Pairs Found from Yeast 3'UTR Sequences

rank	$\|M(\pi, S_f)\|$	$\|M(\pi, S_s)\|$	χ^2 (p-val)	pattern pair
1	55/393	7/379	38.5 ($< 10^{-9}$)	UAAAAAUA $\vee$ UGUAUAA
2	63/393	13/379	34.5 ($< 10^{-8}$)	UAUGUAA $\vee$ UGUAUAA
3	240/393	152/379	33.9 ($< 10^{-8}$)	($\neg$AUCC) $\wedge$ UGUA
4	262/393	174/379	33.8 ($< 10^{-8}$)	($\neg$UAGCU) $\wedge$ UGUA
5	223/393	136/379	33.7 ($< 10^{-8}$)	($\neg$GUUG) $\wedge$ UGUA

TABLE 4
Top Five Scoring Pattern Pairs Found from Human 3'UTR Sequences

rank	$\lvert M(\pi, S)\rvert$	rank sum	avg rank	z (p-val)	pattern pair
1	1338/2306	1.7101×10^6	1278.1	10.56 ($< 10^{-25}$)	UUAUUU $\vee$ UGUAUA
2	904/2306	1.2072×10^6	1335.4	10.53 ($< 10^{-25}$)	UUUUAUUU $\vee$ UGUAUA
3	1410/2306	1.7900×10^6	1269.5	10.49 ($< 10^{-25}$)	UUUAAA $\vee$ UUUAUA
4	711/2306	9.7370×10^5	1369.5	10.40 ($< 10^{-24}$)	UAUUUAU $\vee$ UGUAUAU
5	535/2306	7.5645×10^5	1413.9	10.32 ($< 10^{-24}$)	UGUAAAUA $\vee$ UGUAUAU

where $x = |M(\pi)|$ and $y = R(\pi)$, with appropriate corrections for ranks and variance when there are ties in the decay rate values. The score function can be calculated in constant time for each x and y, provided $O(m \log m)$ time preprocessing for sorting of the data and assigning the ranks.

The top five scoring patterns are presented in Table 4. All pairs are of the form $p \vee q$ common to sequences with higher ranks, that is, sequences with higher decay rates. Notice that most of the highest scoring patterns contain UGUAUA, which was also contained in the results for yeast, which may indicate a possibility that these degradation mechanisms are evolutionarily conserved between eukaryotes. The other pattern in the pairs consists of A and U and apparently captures the A+U rich elements (AREs) [30], which are known to promote rapid mRNA decay dependent on deadenylation. The form $p \vee q$ of the pattern pairs also indicates that the two elements may have *complementary* roles in the degradation of mRNA.

6 Discussion

In this paper, we presented a new formulation of the composite pattern discovery problem: finding *Boolean combinations of patterns*. In contrast to previous composite pattern discovery approaches, our algorithm can find sequence element pairs which may possess competing properties, as well as cooperative ones. We have presented an efficient $O(N^2)$ algorithm for finding the optimal Boolean substring pattern pair with respect to a suitable scoring function from a set of strings that have a numeric attribute value assigned to each string. The algorithm was applied to moderately sized biological sequence data and was successful in finding pattern pairs that captured known destabilizing elements, as well as possible stabilizing elements, from 3'UTR of yeast and human mRNA sequences, where each mRNA sequence is labeled with values depending on its decay rate.

Frequently, in biological applications, motif models which consider ambiguity in the matching are preferred, rather than the "exact" substring patterns used in this paper. Nevertheless, the selection of the motif model for a particular application is still a very difficult problem and substring patterns can be effective, as shown in this paper and others [11]. As well as being efficient, simpler models also have the advantage of being easier to interpret and can be used as a quick, initial scanning for the task.

6.1 Algorithm Variations

6.1.1 Multiple String Attributes

In the previous sections, we assumed that the input consisted of a single set of strings, where each string is paired with a numeric attribute value. The algorithm can be easily modified to account for *two* string attributes and a numeric attribute. Let $S = \{s_1, \ldots, s_m\}$ and $T = \{t_1, \ldots, t_m\}$. For a given pattern pair $\pi = \langle F, p, q\rangle$, we redefine

$$M(\pi) = M(\pi, S, T) = \\ \{i \mid F(\psi(p, s_i), \psi(q, t_i)) = \texttt{true}, s_i \in S, t_i \in T\},$$

that is, p is searched from S, while q is searched from T. Two generalized suffix trees, one for S and the other for T, are constructed: The former is used simply to enumerate the candidates for p, while the latter is used for enumerating q together with the linear-time algorithm for solving the color set size problem. The algorithm would run in $O(N_1^2 + N_1 N_2)$ time and $O(N_1 + N_2)$ space, where $N_1 = \sum_{i=1}^{m} length(s_i)$ and $N_2 = \sum_{i=1}^{m} length(t_i)$. With this change in problem definition, we are able to search for Boolean combinations of patterns from different sequence regions. For example, in the mRNA data sets used previously, if we were to choose the set of 3'UTR sequences of each gene for S and the set of 5'UTR sequences of each gene for T, we could look for possible functional dependencies between sequence elements in the 3'UTR and 5'UTR.

6.1.2 Distance Restrictions

A variation of the problem which considers distance constraints between the occurrences of the two patterns is presented in [35]. Pattern combinations such as $p \wedge_\alpha \neg q$ are considered, which is defined to match a given string s if there exists an occurrence of p in s such that q does NOT occur in s *within* α positions of the occurrence of p, where α is a given integer. The algorithm in this paper is modified to use *sparse* suffix trees and is able to solve the problem optimally for a given α in $O(N^2)$ time.

6.2 Availability

Software that implements the algorithms in this paper is provided at http://bonsai.ims.u-tokyo.ac.jp/~bannai/software/cpd/ under the GNU General Public License.

Acknowledgments

This work was supported in part by Grant-in-Aid for Encouragement of Young Scientists (B) and Grant-in-Aid for Scientific Research on Priority Areas (C) "Genome

Biology" from the Ministry of Education, Culture, Sports, Science, and Technology of Japan. Computational resources for the experiments were provided by The Human Genome Center Super Computer System at the Institute of Medical Science, University of Tokyo. The authors are also grateful to Dr. Seiya Imoto (Human Genome Center, Institute of Medical Science, University of Tokyo) for helpful comments concerning the scoring functions.

References

[1] A. Brazma, I. Jonassen, I. Eidhammer, and D. Gilbert, "Approaches to the Automatic Discovery of Patterns in Biosequences," *J. Computational Biology,* vol. 5, pp. 279-305, 1998.

[2] L. Marsan and M.-F. Sagot, "Algorithms for Extracting Structured Motifs Using a Suffix Tree with an Application to Promoter and Regulatory Site Consensus Identification," *J. Computational Biology,* vol. 7, pp. 345-360, 2000.

[3] H. Arimura, A. Wataki, R. Fujino, and S. Arikawa, "A Fast Algorithm for Discovering Optimal String Patterns in Large Text Databases," *Proc. Int'l Workshop Algorithmic Learning Theory,* pp. 247-261, 1998.

[4] E. Eskin and P.A. Pevzner, "Finding Composite Regulatory Patterns in DNA Sequences," *Bioinformatics,* vol. 18, pp. S354-S363, 2002.

[5] X. Liu, D. Brutlag, and J. Liu, "BioProspector: Discovering Conserved DNA Motifs in Upstream Regulatory Regions of Co-Expressed Genes," *Proc. Pacific Symp. Biocomputing,* pp. 127-138, 2001.

[6] O. Maruyama, H. Bannai, Y. Tamada, S. Kuhara, and S. Miyano, "Fast Algorithm for Extracting Multiple Unordered Short Motifs Using Bit Operations," *Information Sciences,* vol. 146, pp. 115-126, 2002.

[7] S. Shimozono, A. Shinohara, T. Shinohara, S. Miyano, S. Kuhara, and S. Arikawa, "Knowledge Acquisition from Amino Acid Sequences by Machine Learning System BONSAI," *Trans. Information Processing Soc. Japan,* vol. 35, no. 10, pp. 2009-2018, 1994.

[8] A. Shinohara, M. Takeda, S. Arikawa, M. Hirao, H. Hoshino, and S. Inenaga, "Finding Best Patterns Practically," *Progress in Discovery Science,* pp. 307-317, 2002.

[9] M. Takeda, S. Inenaga, H. Bannai, A. Shinohara, and S. Arikawa, "Discovering Most Classificatory Patterns for Very Expressive Pattern Classes," *Proc. Sixth Int'l Conf. Discovery Science,* pp. 486-493, 2003.

[10] D. Shinozaki, T. Akutsu, and O. Maruyama, "Finding Optimal Degenerate Patterns in DNA Sequences," *Bioinformatics,* vol. 19, pp. 206ii-214ii, 2003.

[11] H.J. Bussemaker, H. Li, and E.D. Siggia, "Regulatory Element Detection Using Correlation with Expression," *Nature Genetics,* vol. 27, pp. 167-171, 2001.

[12] H. Bannai, S. Inenaga, A. Shinohara, M. Takeda, and S. Miyano, "A String Pattern Regression Algorithm and Its Application to Pattern Discovery in Long Introns," *Genome Informatics,* vol. 13, pp. 3-11, 2002.

[13] E.M. Conlon, X.S. Liu, J.D. Lieb, and J.S. Liu, "Integrating Regulatory Motif Discovery and Genome-Wide Expression Analysis," *Proc. US Nat'l Academy Sciences,* vol. 100, no. 6, pp. 3339-3344, 2003.

[14] H. Bannai, S. Inenaga, A. Shinohara, M. Takeda, and S. Miyano, "Efficiently Finding Regulatory Elements Using Correlation with Gene Expression," *J. Bioinformatics and Computational Biology,* vol. 2, no. 2, pp. 273-288, 2004.

[15] C.B. -Z. Zilberstein, E. Eskin, and Z. Yakhini, "Using Expression Data to Discover RNA and DNA Regulatory Sequence Motifs," *First Ann. RECOMB Satellite Workshop on Regulatory Genomics,* 2004.

[16] D. Gusfield, *Algorithms on Strings, Trees, and Sequences.* Cambridge Univ. Press, 1997.

[17] Y. Wang, C. Liu, J. Storey, R. Tibshirani, D. Herschlag, and P. Brown, "Precision and Functional Specificity in mRNA Decay," *Proc. US Nat'l Academy of Sciences,* vol. 99, no. 9, pp. 5860-5865, 2002.

[18] E. Yang, E. van Nimwegen, M. Zavolan, N. Rajewsky, M. Schroeder, M. Magnasco, and J. Darnell Jr., "Decay Rates of Human mRNAs: Correlation with Functional Characteristics and Sequence Attributes," *Genome Research,* vol. 13, no. 8, pp. 1863-1872, 2003.

[19] H. Bannai, H. Hyyrö, A. Shinohara, M. Takeda, K. Nakai, and S. Miyano, "Finding Optimal Pairs of Patterns," *Proc. Fourth Int'l Workshop Algorithms in Bioinformatics,* pp. 450-462, 2004.

[20] U. Manber and G. Myers, "Suffix Arrays: A New Method for On-Line String Searches," *SIAM J. Computing,* vol. 22, no. 5, pp. 935-948, 1993.

[21] D.K. Kim, J.S. Sim, H. Park, and K. Park, "Linear-Time Construction of Suffix Arrays," *Proc. 14th Ann. Symp. Combinatorial Pattern Matching,* pp. 186-199, 2003.

[22] P. Ko and S. Aluru, "Space Efficient Linear Time Construction of Suffix Arrays," *Proc. 14th Ann. Symp. Combinatorial Pattern Matching,* pp. 200-210, 2003.

[23] J. Kärkkäinen and P. Sanders, "Simple Linear Work Suffix Array Construction," *Proc. 30th Int'l Colloquium Automata, Languages and Programming,* pp. 943-955, 2003.

[24] T. Kasai, H. Arimura, and S. Arikawa, "Efficient Substring Traversal with Suffix Arrays," Technical Report 185, Dept. of Informatics, Kyushu Univ., 2001.

[25] M.I. Abouelhoda, S. Kurtz, and E. Ohlebusch, "The Enhanced Suffix Array and Its Applications to Genome Analysis," *Proc. Second Int'l Workshop Algorithms in Bioinformatics,* pp. 449-463, 2002.

[26] M.A. Bender and M. Farach-Colton, "The LCA Problem Revisited," *Proc. Latin American Theoretical Informatics,* pp. 88-94, 2000.

[27] S. Alstrup, C. Gavoille, H. Kaplan, and T. Rauhe, "Nearest Common Ancestors: A Survey and a New Distributed Algorithm," *Proc. 14th Ann. ACM Symp. Parallel Algorithms and Architectures,* pp. 258-264, 2002.

[28] L. Hui, "Color Set Size Problem with Applications to String Matching," *Proc. Third Ann. Symp. Combinatorial Pattern Matching,* pp. 230-243, 1992.

[29] T. Kasai, G. Lee, H. Arimura, S. Arikawa, and K. Park, "Linear-Time Longest-Common-Prefix Computation in Suffix Arrays and Its Applications," *Proc. 12th Ann. Symp. Combinatorial Pattern Matching,* pp. 181-192, 2001.

[30] C.J. Wilusz, M. Wormington, and S.W. Peltz, "The Cap-to-Tail Guide to mRNA Turnover," *Nature Reviews: Molecular Cell Biology,* vol. 2, pp. 237-246, 2001.

[31] J. Graber, "Variations in Yeast 3′-Processing Cis-Elements Correlate with Transcript Stability," *Trends in Genetetics,* vol. 19, no. 9, pp. 473-476, http://harlequin.jax.org/yeast/turnover, 2003.

[32] M. Wickens, D.S. Bernstein, J. Kimble, and R. Parker, "A PUF Family Portrait: 3′ UTR Regulation as a Way of Life," *Trends in Genetics,* vol. 18, no. 3, pp. 150-157, 2002.

[33] M.J. Ruiz-Echevarria, R. Munshi, J. Tomback, T.G. Kinzy, and S.W. Peltz, "Characterization of a General Stabilizer Element that Block Deadenylation-Dependent mRNA Decay," *J. Biological Chemistry,* vol. 276, no. 33, pp. 30995-31003, 2001.

[34] A. Kasprzyk, D. Keefe, D. Smedley, D. London, W. Spooner, C. Melsopp, M. Hammond, P. Rocca-Serra, T. Cox, and E. Birney, "EnsMart: A Generic System for Fast and Flexible Access to Biological Data," *Genome Research,* vol. 14, pp. 160-169, 2004.

[35] S. Inenaga, H. Bannai, H. Hyyrö, A. Shinohara, M. Takeda, K. Nakai, and S. Miyano, "Finding Optimal Pairs of Cooperative and Competing Patterns with Bounded Distance," *Proc. Seventh Int'l Conf. Discovery Science,* pp. 32-46, 2004.

Hideo Bannai received the BS and MS degrees in computer science from the University of Tokyo in 1998 and 2000, respectively. He is currently a research associate at the Laboratory of DNA Information Analysis, Human Genome Center, Institute of Medical Science, The University of Tokyo. His current research interests include pattern discovery from biological sequence data.

Heikki Hyyrö received the MS degree in 2000 and the PhD degree in 2003 from the Department of Computer Sciences at the University of Tampere, Finland. He was a postdoctoral research fellow of the Japan Science and Technology Agency during this work, positioned in the Department of Informatics at Kyushu University. His current research interests lie mainly within the general field of string algorithms.

Ayumi Shinohara received the BS degree in 1988 in mathematics, the MS degree in 1990 in information systems, and the Doctor of Sciences degree in 1994, all from Kyushu University. He is now an associate professor in the Department of Informatics at Kyushu University. His current interests include discovery science, machine learning, bioinformatics, and pattern matching algorithms.

Masayuki Takeda received the BS degree in 1987, the MS degree in 1989, and the PhD degree in 1996 from Kyushu University. He is currently a professor in the Department of Informatics, Kyushu University. His current interests include string pattern matching, tree pattern matching, and computational knowledge discovery.

Kenta Nakai received the PhD degree from Kyoto University in 1992. He is now a professor at the Human Genome Center, Institute of Medical Science, University of Tokyo. His research interest is mainly focused on the development of computational methods to interpret genomic sequence data, such as the development of PSORT, a predictor of protein subcellular localization sites.

Satoru Miyano received the BS degree in 1977, the MS degree in 1979, and the PhD degree in mathematics from Kyushu University. He is now a professor at the Human Genome Center, Institute of Medical Science, University of Tokyo. His current interests include computational gene network inference methods, modeling and simulation of biological systems, and computational knowledge discovery. He is on the editorial board of *Bioinformatics,* the *Journal of Bioinformatics and Computational Biology,* and *Theoretical Computer Science* and is the chief editor of *Genome Informatics.*