Finding Optimal Pairs of Patterns

Hideo Bannai[1], Heikki Hyyrö[2], Ayumi Shinohara[2,3], Masayuki Takeda[3], Kenta Nakai[1], and Satoru Miyano[1]

[1] Human Genome Center, Institute of Medical Science,
The University of Tokyo, Tokyo 108-8639, Japan
{bannai,knakai,miyano}@ims.u-tokyo.ac.jp
[2] PRESTO, Japan Science and Technology Agency (JST)
helmu@cs.uta.fi
[3] Department of Informatics, Kyushu University 33, Fukuoka 812-8581, Japan
{ayumi,takeda}@i.kyushu-u.ac.jp

Abstract. We consider the problem of finding the *optimal pair of string patterns* for discriminating between two sets of strings, i.e. finding the pair of patterns that is best with respect to some appropriate scoring function that gives higher scores to pattern pairs which occur more in the strings of one set, but less in the other. We present an $O(N^2)$ time algorithm for finding the optimal pair of *substring patterns*, where N is the total length of the strings. The algorithm looks for all possible Boolean combination of the patterns, e.g. patterns of the form $p \wedge \neg q$, which indicates that the pattern pair is considered to match a given string s, if p occurs in s, AND q does NOT occur in s. The same algorithm can be applied to a variant of the problem where we are given a single set of sequences along with a numeric attribute assigned to each sequence, and the problem is to find the optimal pattern pair whose occurrence in the sequences is *correlated* with this numeric attribute. An efficient implementation based on suffix arrays is presented, and the algorithm is applied to several nucleotide sequence datasets of moderate size, combined with microarray gene expression data, aiming to find regulatory elements that *cooperate, complement, or compete with* each other in enhancing and/or silencing certain genomic functions.

1 Introduction

Pattern discovery from biosequences is an important topic in Bioinformatics, which has been, and is being, studied heavily with numerous variations and applications (see [1] for a survey on earlier work). Although finding the single, most significant pattern conserved across multiple sequences has important and obvious applications, it is known that in many, if not most, actual cases, more than one sequence element is responsible for the biological role of the sequences. There are several methods which address this observation, focussing 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 is another method that looks for composite patterns [4] using *mismatch trees*. Bioprospector [5] applies the Gibbs sampling strategy to find gapped motifs.

The main contribution of this paper is to present an efficient $O(N^2)$ algorithm (where N is the total length of the input strings) and implementation based on suffix arrays, for finding the *optimal pair of substring patterns* combined with any Boolean function. Note that the methods mentioned above for finding composite patterns can be viewed as being limited to using only the $\wedge$ (`AND`) operation. The use of any Boolean function allows the use of the $\neg$ (`NOT`) operation, therefore making it possible to find not only sequence elements that *cooperate* with each other, but those of the form $p \wedge \neg q$, which can be interpreted as two sequence elements with *competing* functions (e.g. positive and negative elements). The pattern pairs discovered by our algorithm are optimal in that they are guaranteed to be the highest scoring pair of 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 sequence sets [6–9], as well as the discovery of patterns that *correlate* with a given numeric attribute (e.g. gene expression level) assigned to the sequences [10–14]. 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 [15], which is an indispensable data structure for efficient processing of substring information. Moreover, the algorithm using the suffix tree can, with the same asymptotic complexity, be simulated very efficiently using suffix arrays, and is thus implemented. 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 [16, 17]. We were successful in obtaining several interesting pattern pairs where some correspond to known mRNA destabilizing elements.

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 i-th 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 de-

Table 1. Summary of candidate Boolean operations on pattern pair $\langle p, F, q \rangle$.

	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`

fine $\langle p, F, q \rangle$ as a *Boolean pattern pair* (or simply *pattern pair*), which consists of two patterns p and q and a Boolean function $F : \{\texttt{true}, \texttt{false}\} \times \{\texttt{true}, \texttt{false}\} \rightarrow \{\texttt{true}, \texttt{false}\}$. The matching function value $\psi(\langle p, F, q \rangle, s)$ 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 $\psi(\pi, s) = \texttt{true}$.

For a given set of strings $S = \{s_1, \ldots, s_m\}$, let $M(\pi, S)$ denote the subset of strings in S that π matches, that is, $M(\pi, S) = \{s_i \in S \mid \psi(\pi, s_i) = \texttt{true}\}$. Now suppose that for each $s_i \in S$, we are given an associated numeric attribute value r_i. Let $\sum_{M(\pi,S)} r_i$ denote the sum of r_i over all s_i such that $\psi(\pi, s_i) = \texttt{true}$, that is, $\sum_{M(\pi,S)} r_i = \sum(r_i \mid \psi(\pi, s_i) = \texttt{true})$. For brevity, we shall omit S where possible and let $M(\pi)$ and $\sum_{M(\pi)} r_i$, be a shorthand for $M(\pi, S)$ and $\sum_{M(\pi,S)} r_i$, respectively.

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 *score* that takes parameters of type $|M(\pi)|$ and $\sum_{M(\pi)} r_i$. The specific choice of the scoring function depends highly 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 $\pi \in \{\langle p, F, q \rangle \mid p, q \in \Sigma^*, F \in \{F_0, \ldots, F_{15}\}\}$ that maximizes $score(|M(\pi)|, \sum_{M(\pi)} r_i)$.

Below, we give examples of choices for *score* and r_i.

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 have some biological function, 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 $\sum_{(\pi,S)} r_i = |M(\pi, S_1)|$. 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 $|S_2|$.

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 [11, 13] is the inter class 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 $y = \sum_{M(\pi)} r_i$. 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* [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 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 [15] with respect to the length of the string (total length of the strings for GST).

A *suffix array* [18] 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, that is $lcp_s[i] = \max\{k \mid s[A_s[i-1]:A_s[i-1]+k-1] = s[A_s[i]:A_s[i]+k-1]\}$. Recently, three methods for constructing the suffix array directly from a string in linear time have been developed [19–21]. The *lcp* array can be constructed from the suffix array also in linear time [22]. 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 [22, 23] (the combination termed in [23] as 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.

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 pre-processed in linear time to answer the lowest common ancestor (*lca-query*) for any given pair of nodes in constant time [24]. In terms of the suffix array, the *lca*-query is almost equivalent to a *range minimum query* (*rm-query*) on the *lcp* array, which, given a pair of positions i and j, $rmq(i, j)$ returns the position of the minimum element in the sub-array $lcp[i:j]$. The *lcp* array can also be pre-processed in linear time to answer the *rm*-query in constant time [24, 25].

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, since they need only be done once, and a naïve algorithm costs $O(N^2)$. 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 $N = \sum_{i=1}^{m} length(s_i)$. We first show that a naïve 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 there are only $O(N)$ 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 $\pi = \langle l(v_1), F, l(v_2)\rangle$, where v_1, v_2 are nodes of the GST, the values $|M(\pi)|$ and $\sum_{M(\pi)} r_i$ can be computed in $O(N)$ time, by using any of the linear time substring matching algorithms.

Then each corresponding scoring function value can be computed in constant time. Therefore, the total time required is $O(N^3)$.

3.2 An $O(N^2)$ Algorithm

Our algorithm is derived from the technique for solving the *color set size problem* [26], 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. The following algorithm computes the values $\sum_{M(l(v))} r_i$ for all v. Note that if we give each attribute r_i the value 1, then $\sum_{M(l(v))} r_i = |M(l(v))|$. 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 $\sum_{LF(v)} r_i = \sum(c_i(v)r_i \mid \psi(l(v), s_i) = \texttt{true})$.

For any node v in the GST over the string set S, the matching value $\psi(l(v), s_i)$ is `true` for at least one string s_i. Thus the equality $\sum_{M(l(v))} r_i = \sum(r_i \mid \psi(l(v), s_i) = \texttt{true}) = \sum_{LF(v)} r_i - \sum((c_i(v) - 1)r_i \mid \psi(l(v), s_i) = \texttt{true})$ holds. Let us define the preceding subtracted sum to be a *correction factor*, which we denote by $corr(l(v), S) = \sum((c_i(v) - 1)r_i \mid \psi(l(v), s_i) = \texttt{true})$.

Since the recurrence $\sum_{LF(v)} r_i = \sum(\sum_{LF(v')} r_i \mid v'$ is a child node of $v)$ clearly holds, the values $\sum_{LF(v)} r_i$ can be easily calculated for all v during a linear time bottom-up 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 depth-first 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 $s_i \in S$, $\psi(l(v), s_i) = \texttt{true}$ 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 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 $(c_i(v) - 1)r_i$. 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 $\sum((c_i(v) - 1)r_i \mid \psi(l(v), s_i) = \texttt{true}) = corr(l(v), S)$. Hence at this point a single linear time bottom-up 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 $\sum_{M(l(v))} r_i$.

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 $\sum_{M(l(v))} r_i$ is linear.

The above described algorithm permits us to compute the values $\sum_{M(l(v))} r_i$ and $|M(l(v))|$ in linear time, which in turn leads into a linear time solution for the problem of finding a good pattern when the pattern is a *single* substring: The scoring function can now be computed for each possible pattern candidate $l(v)$. Also the case of a Boolean pattern pair will be solved in this manner. That is, we will concentrate on how to compute the values $\sum_{M(\pi)} r_i$ (and $|M(\pi)|$) for all possible $O(N^2)$ pattern pair candidates, where $\pi = \langle l(v_1), F, l(v_2)\rangle$ 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.

Naïve 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, since computing the needed values $\psi(\langle l(v_1), F, l(v_2)\rangle, s_1)$ requires us to somehow conduct an intrinsic set operation between the string subsets that match / 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.

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 in order to process the $O(N)$ choices for the second pattern, $l(v_2)$, in $O(N)$ time. 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. 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 the values $\sum_{M(l(v)),\mathtt{b}} r_i = \sum(r_i \mid \psi(l(v), s_i) = \mathtt{true} \wedge \psi(l(v_1), s_i) = \mathtt{b})$ for all nodes in linear time. We note that $\sum(r_i \mid \psi(l(v), s_i) = \mathtt{false} \wedge \psi(l(v_1), s_i) = \mathtt{b}) = \sum(r_i \mid \psi(l(v_1), s_i) = \mathtt{b}) - \sum_{M(l(v)),\mathtt{b}} r_i$, where the value $\sum(r_i \mid \psi(l(v_1), s_i) = \mathtt{b})$ can easily be computed in linear time. Thus *all* cumulative values of form $\sum(r_i \mid \psi(l(v), s_i) = \mathtt{b_1} \wedge \psi(l(v_1), s_i) = \mathtt{b_2})$, where $\mathtt{b_1}, \mathtt{b_2} \in \{\mathtt{true}, \mathtt{false}\}$, can be computed in linear time. And from these it is straightforward to compute the values $\sum_{M(\langle l(v_1), F, l(v_2)\rangle)} r_i = \sum(r_i \mid F(\psi(l(v_1), s_i), \psi(l(v_2), s_i)) = \mathtt{true})$, as well as the corresponding scoring function values, in linear time. Thus, given a fixed $l(v_1)$, we can compute the scores for all pattern pair candidates of form $\langle l(v_1), F, l(v_2)\rangle$ 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.

It is not difficult to see that the space complexity of the algorithm is $O(N)$. The algorithm is also highly parallelizable, since the $O(N)$ time calculations for each fixed $l(v_1)$ are independent of each other.

4 Implementation Using Suffix Arrays

The algorithm on the suffix tree can be simulated efficiently by a suffix array. We modify the algorithm of [22, 27] that simulates a bottom-up traversal of a suffix tree, using a suffix array. Notice that since each suffix of the string corresponds

to a leaf in the suffix tree, each position in the suffix array corresponds to a leaf in the suffix tree. A subtlety in the modification lies in determining where to store the correction factors after calculating the lca, since the simulation via suffix arrays does not explicitly recreate the internal nodes of the suffix tree. For storing the correction factors, we construct another array C of the same length as the suffix array, which represents the internal nodes of the suffix tree. An element $C[i]$ in the array corresponds to the lca of suffix $A_S[i-1]$ and suffix $A_S[i]$. When adding up the correction factor for two leaves corresponding to $A_S[i]$ and $A_S[j]$ $(i < j)$, the correction factor is added into $C[rmq(i+1, j)]$. Although it is the case that different positions in C can correspond to the same internal node when the node has more than two children, it is possible to correctly sum the values required for the score calculations, since all positions of the array are visited in the traversal simulation, and the addition operation on numeric values is commutative as well as associative.

5 Computational Experiments

The degradation of mRNA, in addition to transcription, is one of several important mechanisms which control the expression level of a gene (see [28] 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 [16, 17]. 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.

The algorithm was implemented using the C language, and uses POSIX threads to execute parallel computations. To give an idea of the efficiency of the algorithm, it can be run on a data set with $720,673$ candidates for a single pattern, meaning $720,673^2 = 519,369,572,929$ possible pattern pairs, requiring about half a day on a Sun Fire 15K with 96 CPUs (UltraSPARC III Cu 1.2GHz). To ease the interpretation process in the following experiments, we limit the search to Boolean combinations: $p' \wedge q'$ and $p' \vee q'$, where p' is either p or $\neg p$, and q' is either q or $\neg q$.

5.1 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 [29], which are constructed based on the microarray experiments in [16] 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

Table 2. Top 5 scoring pattern pairs found from yeast 3'UTR sequences.

rank	$\lvert M(\pi, S_f)\rvert$	$\lvert M(\pi, S_s)\rvert$	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

traversal on the suffix array on this dataset 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 chi-squared statistic, calculated by $(|S_f| + |S_s|)(\mathtt{tp} * \mathtt{tn} - \mathtt{fp} * \mathtt{fn})^2/(\mathtt{tp} + \mathtt{fn})(\mathtt{tp} + \mathtt{fp})(\mathtt{tn} + \mathtt{fp})(\mathtt{tn} + \mathtt{fn})$ where $\mathtt{tp} = |M(\pi, S_f)|$, $\mathtt{fp} = |S_f| - \mathtt{tp}$, $\mathtt{tn} = |S_s| - \mathtt{fn}$, and $\mathtt{fn} = |M(\pi, S_s)|$.

The time required for computation was around $3 \sim 4$ minutes on the above mentioned computer. The 5 top scoring pattern pairs found are shown in Table 2. 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 [30], and has also been found in the previous analysis [29] to be significantly over-represented 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 [31]. 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 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 [17]. 3'UTR sequences for each mRNA was retrieved using the ENSMART [32] interface. We were able to obtain 2306 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 statistic. 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 $\sum_{M(\pi)} r_i$ approximately depends on the

Table 3. Top 5 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

normal distribution when the sample size is large. Therefore, we use the z-score defined by: $z(x, y) = \frac{(y - x(|S|+1)/2)}{\sqrt{x(|S|-x)(|S|+1)/12}}$ where $x = |M(\pi)|$ and $y = \sum_{M(\pi)} r_i$, 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 5 top scoring patterns are presented in Table 3. 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 contained also 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 consist of A and U, and apparently captures the A+U rich elements (AREs) [28] 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

We have presented an efficient $O(N^2)$ algorithm for finding the optimal Boolean pattern pair with respect to a suitable scoring function, from a set of strings and 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 [10]. 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.

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. The authors would like to acknowledge Dr. Seiya Imoto (Human Genome Center, Institute of Medical Science, University of Tokyo) for helpful comments concerning the scoring functions.

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. Marsan, L., Sagot, M.F.: Algorithms for extracting structured motifs using a suffix tree with an application to promoter and regulatory site consensus identification. J. Comput. Biol. **7** (2000) 345–360
3. Arimura, H., Wataki, A., Fujino, R., Arikawa, S.: A fast algorithm for discovering optimal string patterns in large text databases. In: International Workshop on Algorithmic Learning Theory (ALT'98). Volume 1501 of LNAI., Springer-Verlag (1998) 247–261
4. Eskin, E., Pevzner, P.A.: Finding composite regulatory patterns in DNA sequences. Bioinformatics **18** (2002) S354–S363
5. Liu, X., Brutlag, D., Liu, J.: BioProspector: discovering conserved DNA motifs in upstream regulatory regions of co-expressed genes. In: Pac. Symp. Biocomput. (2001) 127–138
6. Shimozono, S., Shinohara, A., Shinohara, T., Miyano, S., Kuhara, S., Arikawa, S.: Knowledge acquisition from amino acid sequences by machine learning system BONSAI. Transactions of Information Processing Society of Japan **35** (1994) 2009–2018
7. 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
8. 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
9. Shinozaki, D., Akutsu, T., Maruyama, O.: Finding optimal degenerate patterns in DNA sequences. Bioinformatics **19** (2003) 206ii–214ii
10. Bussemaker, H.J., Li, H., Siggia, E.D.: Regulatory element detection using correlation with expression. Nature Genetics **27** (2001) 167–171
11. 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
12. Conlon, E.M., Liu, X.S., Lieb, J.D., Liu, J.S.: Integrating regulatory motif discovery and genome-wide expression analysis. Proc. Natl. Acad. Sci. **100** (2003) 3339–3344
13. 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 (2004) in press.
14. 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)
15. Gusfield, D.: Algorithms on Strings, Trees, and Sequences. Cambridge University Press (1997)

16. Wang, Y., Liu, C., Storey, J., Tibshirani, R., Herschlag, D., Brown, P.: Precision and functional specificity in mRNA decay. Proc. Natl. Acad. Sci. **99** (2002) 5860–5865
17. Yang, E., van Nimwegen, E., Zavolan, M., Rajewsky, N., Schroeder, M., Magnasco, M., Jr., J.D.: Decay rates of Human mRNAs: correlation with functional characteristics and sequence attributes. Genome Res. **13** (2003) 1863–1872
18. Manber, U., Myers, G.: Suffix arrays: a new method for on-line string searches. SIAM Journal on Computing **22** (1993) 935–948
19. Kim, D.K., Sim, J.S., Park, H., Park, K.: Linear-time construction of suffix arrays. In: 14th Annual Symposium on Combinatorial Pattern Matching (CPM 2003). Volume 2676 of LNCS., Springer-Verlag (2003) 186–199
20. Ko, P., Aluru, S.: Space efficient linear time construction of suffix arrays. In: 14th Annual Symposium on Combinatorial Pattern Matching (CPM 2003). Volume 2676 of LNCS., Springer-Verlag (2003) 200–210
21. Kärkkäinen, J., Sanders, P.: Simple linear work suffix array construction. In: 30th International Colloquium on Automata, Languages and Programming (ICALP 2003). Volume 2719 of LNCS., Springer-Verlag (2003) 943–955
22. Kasai, T., Arimura, H., Arikawa, S.: Efficient substring traversal with suffix arrays. Technical Report 185, Department of Informatics, Kyushu University (2001)
23. Abouelhoda, M.I., Kurtz, S., Ohlebusch, E.: The enhanced suffix array and its applications to genome analysis. In: Second International Workshop on Algorithms in Bioinformatics (WABI 2002). Volume 2452 of LNCS., Springer-Verlag (2002) 449–463
24. Bender, M.A., Farach-Colton, M.: The LCA problem revisited. In: Proc. Latin American Theoretical Informatics (LATIN). Volume 1776 of LNCS., Springer-Verlag (2000) 88–94
25. Alstrup, S., Gavoille, C., Kaplan, H., Rauhe, T.: Nearest common ancestors: a survey and a new distributed algorithm. In: 14th annual ACM symposium on Parallel algorithms and architectures. (2002) 258–264
26. 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
27. 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
28. Wilusz, C.J., Wormington, M., Peltz, S.W.: The cap-to-tail guide to mRNA turnover. Nat. Rev. Mol. Cell Biol. **2** (2001) 237–246
29. Graber, J.: Variations in yeast 3'-processing cis-elements correlate with transcript stability. Trends Genet. **19** (2003) 473–476 `http://harlequin.jax.org/yeast/turnover/`.
30. Wickens, M., Bernstein, D.S., Kimble, J., Parker, R.: A PUF family portrait: 3'UTR regulation as a way of life. Trends Genet. **18** (2002) 150–157
31. Ruiz-Echevarria, M.J., Munshi, R., Tomback, J., Kinzy, T.G., Peltz, S.W.: Characterization of a general stabilizer element that blocks deadenylation-dependent mRNA decay. J. Biol. Chem. **276** (2001) 30995–31003
32. Kasprzyk, A., Keefe, D., Smedley, D., London, D., Spooner, W., Melsopp, C., Hammond, M., Rocca-Serra, P., Cox, T., Birney, E.: EnsMart: A generic system for fast and flexible access to biological data. Genome Research **14** (2004) 160–169