A Fast Algorithm for Discovering Optimal String Patterns in Large Text Databases

Arimura, Hiroki
Department of Informatics, Kyushu University

Wataki, Atsushi
Department of Informatics, Kyushu University

Fujino, Ryoichi
Department of Informatics, Kyushu University

Arikawa, Setsuo
Department of Informatics, Kyushu University

出版情報：DOI Technical Report. 148, 1998-03-19. Department of Informatics, Kyushu University
バージョン：
権利関係：

DOI-TR-xxx

DOI Technical Report

A Fast Algorithm for Discovering Optimal String Patterns in Large Text Databases

by

H. Arimura, A. Wataki, R. Fujino, S. Arikawa

March 19, 1998

Department of Informatics
Kyushu University
Fukuoka 812-81, Japan

Email: arim@i.kyushu-u.ac.jp   Phone: +81-92-583-7632

A Fast Algorithm for Discovering Optimal String Patterns in Large Text Databases

Hiroki Arimura, Atsushi Wataki, Ryoichi Fujino, Setsuo Arikawa

Department of Informatics, Kyushu University
Kasuga Koen, Kasuga 816, Japan
{arim,wataki,fujino,arikawa}@i.kyushu-u.ac.jp
http://robin.i.kyushu-u.ac.jp/~arim/

Abstract

We consider a data mining problem in a large collection of unstructured texts based on association rules over subwords of texts. A two-word association pattern is an expression such as

$$(\texttt{TATA, 30, AGGAGGT}) \Rightarrow C$$

that expresses a rule that if a text contains a subword `TATA` followed by another subword `AGGAGGT` with distance no more than 30 letters then a property C will hold with a probability. We present an efficient algorithm for computing frequent patterns (α, k, β) that optimize the confidence with respect to a given collection of texts. The algorithm runs in time $O(mn^2)$ and space $O(kn)$, where m and n are the number and the total length of classification examples, respectively, and k is a small constant around $30 \sim 50$. Furthermore for most random and nearly random texts like DNA sequences, the algorithm runs very efficiently in time $O(kn \log^2 n)$. Thus, this algorithm is much faster than a straightforward algorithm that enumerates all the possible patterns in time $O(n^5)$. We also discuss some heuristics such as sampling and pruning for practical improvement. Then, we evaluate the efficiency and the performance of the algorithm with experiments on genetic sequences.

1 Introduction

The recent progress of communication and network technologies, e.g., electronic mail, World Wide Web, and inter/intra networks make it easy for computer users to accumulate a large collection of unstructured or semi-structured texts on their computers at a low cost (Abiteboul 1997). Such *text databases* may be collections of web pages or SGML documents (OPENTEXT Index 1997), protein databases in molecular biology (GenBank 1997), online dictionary (Gonnet 1987), or plain texts on a file system.

There has been a potential demand for efficient discovery of useful information from text databases beyond the power of the present access methods in information retrieval (Abiteboul 1997; Lewis 1996; Wang *et al.* 1994). However, the present data mining technologies are not directly applicable to those unstructured text data because they mainly deal with well-structured data such as relational databases with Boolean or numeric attributes (Agrawal *et al.* 1993; Fukuda *et al.* 1996; Han *et al.* 1992). The difficulties arise for text databases because

- The amount of data is huge, which typically ranges from mega bytes (10^6) in private collections to tera bytes (10^{12}) in web databases.
- A text databases is simply a collection of unstructured string of letters. Thus, the number of possible primitive attributes such as the keywords and the subwords of texts is quite large.

There have been many proposals in data mining from text and sequence data (Feldman and Dagan 1995; Lewis 1996; Mannila *et al.* 1995; Mannila *et al.* 1996; Motowani *et al.* 1996; Wang *et al.* 1994). In this paper, we consider a very simple class of rules called word association patterns and give efficient algorithms for discovering interesting patterns from a large collection of unstructured strings. Adopting the framework recently proposed by Fukuda, Morimoto, Morishita, and Tokuyama (1996), we develop a fast and robust discovery method.

In this paper, we consider the discovery of simple patterns called two-word association patterns. Given a collection S of texts and an objective condition C over S, a *two-word association pattern* or *proximity pattern* is an expression of the form

$$(\texttt{TATA, 30, AGGAGGT}) \Rightarrow C$$

that represent a rule that if a text contains a subword `TATA` followed by another subword `AGGAGGT` with distance no more than 30 letters then the objective condition C will hold with a probability. For simplicity, we omit the objective condition C. This class of rules is a very restricted subclass of the rules studied in (Wang *et al.* 1994) but is considered to be useful for applications such as bioinformatics (Gras and Nicolas 1996), bibliographic search (OED 1987), and Web search (OPENTEXT Index 1997).

As the framework of discovering patterns, we adopt the *optimal rule discovery* (Fukuda, Morimoto, Morishita, and Tokuyama 1996). A *sample* is a finite set $S = \{s_1, \ldots, s_m\}$ of strings. Each elements s_i of S is called a *document* for $1 \leq i \leq m$. An *objective condition* over S is a binary labeling function $C : S \to \{0, 1\}$ A document s is said to be *positive* if $C(s) = 1$ and *negative* otherwise.

For a pattern P, we define $match_S(P)$ and $hit_S(P)$ as the number of the documents and the number of the positive documents, respectively, that P matches. Then, the *support* of P, denoted by $supp_S(P)$, is defined by the percentage of the positive documents in S that P matches to the all documents, that is, $supp_S(P) = hit_S(P)/card(S)$. Intuitively, the support represents the *utility* of a pattern P. Given a parameter $0 \leq \sigma \leq 1$ called the *minimum support*, a pattern P is said to be *frequent* if $supp_S(P) \geq \sigma$. The *confidence* of

P, denoted by $conf_S(P)$, is the percentage of the positive documents in S that P matches to the documents that P matches, that is, $conf_S(P) = hit_S(P)/match_S(P)$.

We state the *optimized confidence pattern problem* as follows: Given a sample set S, an objective condition C, constant k and σ, find all/some frequent patterns $P = (\alpha, k, \beta)$ that optimizes the confidence with respect to S. We can generalize the problem by replacing the confidence $conf_S$ with a certain criterion G_S. Our algorithm scheme also works for other criteria than the confidence such as the length maximization with $G_S(P) = |\alpha|+|\beta|$ (Wang *et al.* 1996) and the classification error minimization with $G_S(P) = \sum_{s \in S}[P(s) \neq C(s)]$ (Maass 1994) [1]. The consistency problems, e.g., Nakanishi *et al.* (1995), are weaker variant of the error minimization.

The optimized confidence patterns can be computed in time $O(n^5)$ by a straightforward algorithm that enumerates $O(n^4)$ possible two-word association patterns since there are at most possible $O(n^2)$ subwords of A. However, this polynomial is too large to apply this algorithm to real applications.

To the problem, in Section 3, we first present an algorithm that computes all the two-word association patterns (α, k, β) using data structures from string matching and computational geometry, the suffix tree and the orthogonal range query. The idea is to reduce the discovery of patterns to that of axes-parallel rectangles over the 2-dimensional plane of suffix ranks. This algorithm runs in time $O(mn^2 \log^2 n)$ and in space $O(kmn \log n)$. where m and n are the number and the total size of texts, respectively, and k is a proximity. Next in Section 4, implementing the orthogonal range queries directly over the suffix tree, we give a modified version of the algorithm that runs in time $O(mn^2)$ in the worst case and $O(kn \log^2 n)$ on nearly random texts like DNA sequences. In Section 5, we introduce some heuristics and examine their performances. Finally in Section 6, we evaluate the efficiency and the performance of our algorithm with experiments on generic sequence from GenBank databases.

2 Preliminaries

2.1 Texts and patterns

For a set S, $card(S)$ denotes the number of elements in S. For nonnegative integers i, j, $[i..j]$ denotes the interval $\{i, i+1, \ldots, j\}$ if $i \leq j$ and $\emptyset$ otherwise.

Let Σ be a finite alphabet of letters. For a string s and a set S of strings, we denote by $|s|$ and by $size(S)$ the length of s and the total length of the strings in S. Let $s = a_1 a_2 \cdots a_n \in \Sigma^*$ be a text of length n. A *position* is any positive integer $1 \leq p \leq n$. If there exist some $u, v, w \in \Sigma^*$ such that $t = uvw$ then we say that u, v and w are a *prefix*, a *subword* and a *suffix* of t, respectively. For positions i, j $(i \leq j)$, we denote by $s[i..j]$ the *subword of s starting at position i and ending at position j*, that is, $a_i a_{i+1} \cdots a_j$.

A *text* is any string A over Σ. A *two-word association pattern* is an expression of the form (α, k, β), where $\alpha, \beta \in \Sigma^*$ are strings over Σ and $k \geq 0$ be a nonnegative integer. For a string $\alpha \in \Sigma^*$, if $\alpha = A[p..p + |\alpha| - 1]$ for some p then we say p is an occurrence of α in A. For a pattern $P = (\alpha, k, \beta)$ if p and q are the occurrences of α and β in A for some (p, q), respectively, and if $0 \leq q - p \leq k$ then we say (p, q) is an *occurrence* of P. We denote the set of all the occurrences of the pattern P in A by $Occ_A(P)$.

[1] $P(s) \in \{0, 1\}$ is 1 iff P occurs in s. This problem plays an essential role in computational learning theory with noise (Kearns,Shapire, Sellie 1994).

2.2 Suffix trees

A suffix tree is a data structure for storing all subwords of a given text in very economical way (McCreight1976). Let $A = a_1 a_2 \cdots a_{n-1}\$$ be a text of length n. We assume that the text always terminates with a special symbol $\$ \notin \Sigma$ distinct from any letter including itself. For each $1 \leq p \leq n$, we define the suffix starting at position p by $A_p = a_p \cdots a_{n-1}\$$.

Then, the *suffix tree* for text A is exactly the *compact trie* for all the suffixes of A, that is, obtained from a trie for A by iteratively removing the internal nodes with only one child and merging the labels of the removed edges.

More precisely, the suffix tree for A is a rooted tree $Tree_A$ that satisfies the following conditions. (i) Each edge is labeled by a subword α of A, which is encoded by a pair (p, q) of positions that points an occurrence of α in A, that is, $A[p, q] = \alpha$. (ii) The labels of any two edges leaving from the same node start with mutually *distinct* letters. (iii) Each node v represents the string $Word(v)$ obtained by concatenating the labels on the path from the root to v in this order. (iv) For $1 \leq i \leq n$, the i-th leaf l_i represents the suffix of rank i in the lexicographic order over all the suffixes of A.

From (iv) and (iii) above $Tree_A$ has exactly n leaves and at most $n-1$ internal nodes, and thus from (i) it requires $O(n)$ space representing $O(n^2)$ subwords of A. Furthermore, McCreight (1976) gives an elegant algorithm that computes $Tree_A$ in linear time and space. It is known that the average height of a suffix tree for random is $O(\log n)$. This is also the case for genetic sequences.

2.3 Orthogonal range query

Let n be a positive integer. Assume that we are given a finite collection X of points over a discrete two-dimensional plane $[1..n] \times [1..n]$. An *orthogonal range query* is to find all the points in X that are included in a given rectangle $[x_1..x_2] \times [y_1..y_2]$. Several solutions have been proposed for the problem, and among them we adopt the method described in Preparata and Shamos (1985) for its simplicity although it is not optimum in computation time. Their solution uses a data structure called the *orthogonal range tree* that requires $O(m \log m)$ space, $O(m \log m)$ preprocessing time, and $O(\log^2 m)$ time per query , where m is the number of points in X. For the algorithm in Section 4, we extends this data structure to search over the suffix tree.

3 The Mining Algorithm

In this section, we first show that there exists an efficient algorithm that computes optimized confidence patterns in time $O(mn^2 \log^2 n)$ and space $O(kmn \log n)$ using the suffix tree and the orthogonal range tree as its data structures. Then, in the next section, we show that we can make orthogonal range queries directly over the suffix tree instead of the range tree. This yields a faster algorithm for the optimized confidence pattern problem.

Figure 1 shows our data mining algorithm $Find\_Optimal$, which finds the optimized confidence patterns in canonical form using an equivalence relation $\equiv_A$ over patterns. The keys of this algorithm are a step to enumerate representative patterns and another step to compute $supp(P)$ and $conf(P)$ quickly. We will describe the details of efficient implementations of these steps in the following subsections.

Procedure: $Find\_Optimal$;
Given: a sample $S = \{s_1, \ldots, s_m\}$, the objective condition C,
the minimum support $0 \leq \sigma \leq 1$, and the proximity $k \geq 0$.
Output: the optimized confidence patterns (α, k, β) in canonical form.
Variable: an orthogonal range tree D and a priority queue Q.

begin
  $D := \emptyset$; $Q := \emptyset$;
  transform S into $A = t_1\$ \cdots \$t_m\$$; compute C and doc over A;
  compute the suffix tree $Tree_A$ for A and suffix arrays suf, pos.
  $INTER_k := \{\ (p, q) \mid 1 \leq p,\ q \leq n,\ 0 \leq (q - p) \leq k,\ doc(p) = doc(q)\ \}$;
  Foreach $(p, q) \in INTER_k$ **do**
    insert $(pos(p), pos(q), doc(p))$ into D;
  Foreach node u in $Tree_A$ **do** /* traversing $Tree_A$ from the leaves to the root */
    compute $L(u), R(u)$ recursively;
  Foreach node u in $Tree_A$ **do** /* traversing $Tree_A$ from the root to the leaves */
    Foreach node v in $Tree_A$ **do** /* traversing $Tree_A$ from the root to the leaves */
      $P := (Word(u), k, Word(v))$;
      make an orthogonal range query $[L(u), R(u)] \times [L(v), R(v)]$ for D;
      compute $supp(P)$ and $conf(P)$ from the result of the query;
      if $supp(P) \geq \sigma$ **then** insert P into the priority queue Q with the key $conf(P)$;
    end;
  end;
  output all the patterns $P \in Q$ that has the highest confidence $conf(P)$;
end

Figure 1: An algorithm for discovering the optimized confidence patterns

3.1 Enumerating the representative patterns using a suffix tree

First, we define an equivalence relation $\equiv_A$ over word association patterns induced from the suffix tree for the sample S. Let $S = \{t_1, \ldots, t_m\}$ be a set of m strings and $C : S \rightarrow \{0, 1\}$ be an objective condition over S.

To extend the suffix tree for sets of texts, we transform a set of texts into a single text as follows. Given S as input, our algorithm merges all texts in S into a single text $A = t_1\$ \cdots \$t_m\$$ by concatenating these texts delimited with an endmarker $\$ \notin \Sigma$, which is a special symbol distinct from any letter including itself. We also define the objective condition C and the document index doc over the positions in A as follows. For each position p, if i-th text $t_i \in S$ includes p then we define $doc(p) = i$ and $C(p) = C(t_i)$. In what follows, we refer to the string A associated with C and doc as the *input text* and denote the length of A by $n = |A|$.

Next, we build the suffix tree $Tree_A$ for the obtained text A in linear time and space by using the suffix tree construction algorithm of McCreight (1976). It is easy to see that the tree $Tree_A$ is isomorphic to the compacted trie [2] for all the suffixes appearing in the original sample S except the labels of the edges directed to the leaves (Amir *et al.* 1994).

Now, we introduce an equivalence relation $\equiv_A$ as follows. For a string α, we define

[2]The compacted trie for the suffixes of a set of texts is also called a *generalized suffix tree* (GST) (Wang *et al.* 1994). The construction here is actually a standard method to build GST in linear time (Amir *et al.* 1994).

$Occ_A(\alpha)$ to be the set of all occurrences of α in A, where $\alpha, \alpha_1, \alpha_2, \beta, \beta_1, \beta_2 \in \Sigma^*$.

- For strings, $\alpha \equiv_A \beta$ iff $Occ(\alpha) = Occ(\beta)$.
- For patterns, $(\alpha_1, k, \beta_1) \equiv_A (\alpha_2, k, \beta_2)$ iff $\alpha_1 \equiv_A \alpha_2$ and $\beta_1 \equiv_A \beta_2$.

If $P \equiv_A Q$ then we say P and Q are *equivalent*. It is easy to see that the next lemma holds for $\equiv_A$.

Lemma 1 *Equivalent patterns give the same value for $supp_S(P)$ and $conf_S(P)$.*

Proof: By the definition of $\equiv_A$, equivalent patterns have the same set of the occurrences in A. Therefore, equivalent patterns P occur in the same set of documents in S and give the same value in $match_T(P)$ for any subset $T \subseteq S$. Since $supp_S(P)$ and $conf_S(P)$ are defined with $match_S(P)$, the result immediately follows. □

From Lemma 1 above, we know that it is sufficient to consider all the representatives with respect to $\equiv_A$ to find an optimized pattern. To enumerate such representatives, we use the suffix tree. Let α be a subword of A. The *locus* of α in $Tree_A$, denoted by $Locus(\alpha)$, is the unique node v of $Tree_A$ such that α is a prefix of $Word(v)$ and $Word(Parent(v))$ is a proper prefix of α, where $Parent(v)$ denotes the parent of v. Since every subword α appearing in A has its unique locus, we define $Rep(\alpha) = Word(Locus(\alpha))$.

Lemma 2 *For any pattern (α, k, β), $(Rep(\alpha), k, Rep(\beta)) \equiv_A (\alpha, k, \beta)$.*

Proof: Let v be a node of $Tree_A$ and $subtree(v)$ be the subtree of $Tree_A$ with root v. For $1 \leq p \leq n$, the subword $Word(v)$ appears in A at p iff $subtree(v)$ includes the leaf representing suffix A_p. Suppose now that we have the uncompacted version of trie $\hat{Tree}_A$ for A and map the nodes in $Tree_A$ into those in $\hat{Tree}_A$. Then, we can easily see that for any node v, v and $Rep(v)$ are mapped on the same edge in $\hat{Tree}_A$. Thus, $subtree(v)$ and $subtree(Rep(v))$ have the same set of leaves. Therefore, we have $Rep(\alpha) \equiv_A \alpha$ for all string α, and hence, the result follows. □

A pattern is said to be in *canonical form* if it has the form $(Word(u), k, Word(v))$ for some nodes u, v of $Tree_A$. Since every pattern that occurs in A at least once has its unique canonical version, we have the following lemma.

Lemma 3 *The set of the canonical patterns forms a set of the representatives of all patterns with respect to $\equiv_A$. Furthermore, the number of such representatives is $O(n^2)$ for any fixed proximity $k \geq 0$.*

3.2 Computing the support values using range queries

In this section, we show that the support and the confidence can be quickly computable by making orthogonal range queries. The technique used here is basically due to Manber and Baeza-Yates (1991). In the next section, we will further extend this technique to improve the time and space complexity of our algorithm.

Suppose that there exists some ordering over letters, and that we arrange all the suffices of A in the lexicographic order over Σ^*. Let $A_{p_1}, A_{p_2}, \ldots, A_{p_n}$ be the obtained sequence, where A_p is the suffix of A starting at position p. Then, we store the indexes $p_1, p_2, \ldots, p_n$ in an array $suf : [1..n] \rightarrow [1..n]$ of length n in this order, and define the array $pos : [1..n] \to [1..n]$ as the inverse mapping of suf. These arrays are called the *suffix array* (Manber and Baeza-Yates 1991). By definition, $suf(i)$ is the position of the suffix of rank i and $pos(p)$ is the rank of the suffix A_p. It is most important in the suffix array that for any subword α of A, the suffixes that have prefix α in common occupy a contiguous maximal subinterval, denoted by $I(\alpha)$, in array suf.

The idea is to reduce the problem of discovering optimized patterns to that of discovering axes-parallel rectangles over 2-dimensional plane. The first step is to transform a pair (p, q) of positions in input text A into a point in 2-dimensional plane $[1..n] \times [1..n]$, called *position space*. Let $INTER_k$ be a diagonal of width k, that is,

$$INTER_k = \{ (p, q) \mid 1 \leq p,\ q \leq n,\ 0 \leq (q - p) \leq k,\ doc(p) = doc(q). \}$$

Then, we transform the points in $INTER_k$ from the position space to the *rank space* as follows:

$$R_k = \{ (pos(p), pos(q)) \mid (p, q) \in INTER_k \}.$$

Now, we associate with a pattern (α, k, β) an axis-parallel rectangle $I(\alpha) \times I(\beta)$ as follows.

Lemma 4 *For any pair $(p, q) \in [1..n] \times [1..n]$,*

- *The pattern (α, k, β) occurs in A at position (p, q), iff*
- *The point $(pos(p), pos(q))$ is a member of R_k, and the axis-parallel rectangle $(I(\alpha) \times I(\beta))$ includes $(pos(p), pos(q))$.*

Assume that each pair (p, q) of positions is labeled with the name of the document that includes the points, that is, $doc(p) = doc(q)$. Then, from Lemma 4, the problems of computing $match(P)$ or $hit(P)$ reduces to the problem of computing the set of, or at least the number of, distinct labels of points included by a given rectangle. From (Preparata and Shamos 1985) and a discussion in Section 2.2.3 a standard argument show the following lemma.

Lemma 5 (Preparata and Shamos 1985) *Let $X \subseteq [1..N] \times [1..N]$ be a set of points labeled by integers $1 \leq l \leq m$. The problem of computing the distinct labels of points in X that are included in a given rectangle $[x_1..x_2] \times [y_1..y_2]$ is solvable in time $O(m \log^2 n)$ and space $O(mn \log n)$ with preprocessing time $O(n \log n)$, where m is the maximum number of distinct labels and n is the number of points in X.*

Theorem 6 *Let S be a sample, C be an objective condition over C, $k \geq 0$ and $0 \leq \sigma \leq 1$ be fixed constants. Then, algorithm Find_Optimal in Figure 1 computes all the optimized confidence patterns in canonical form with proximity k and support threshold σ in time $O(mn^2 \log^2 n)$ and space $O(kmn \log n)$, where $m = card(S)$ and $n = size(S)$.*

Proof: First we build the suffix tree $Tree_A$ in linear time and space. Then, compute intervals $I(v)$ for all node v in time $O(n)$ with dynamic programming (Preparata and Shamos 1985). From Lemma 1 and Lemma 2, it suffices to search at most $O(n^2)$ canonical patterns $P = (Word(u), k, Word(v))$ by enumerating a pair u, v of nodes of $Tree_A$. Then, we can see from Lemma 4 and Lemma 5 that for each P, we can compute $supp(P)$ and $conf(P)$ in $O(kmn \log n)$ preprocessing time, $O(kmn \log n)$ space, and $O(m \log^2 n)$ time per query. Since the number of possible patterns in canonical form is $O(n^2)$, this proves the result. □

4 Modified algorithm

In this section, we present a modified version of our algorithm, that runs in time $O(mn^2)$ and space $O(\max\{k, m\}n)$, In the algorithm, we implement an orthogonal range query mechanism over the suffix tree itself instead of a range tree. Figure 2 shows a modified version of our algorithm. Although a suffix tree is not a balanced tree in general, we can show the following theorem using a technique in Maass (1994).

Procedure: $Modified\_Find\_Optimal$;

Given: a sample $S = \{s_1, \ldots, s_m\}$, the objective condition C, the minimum support $0 \leq \sigma \leq 1$, and the proximity $k \geq 0$.

Output: the optimized confidence patterns (α, k, β) in canonical form.

Variable: a priority queue Q, for each node u the lists $B(u) = \{\langle x, y, z\rangle\}$ and $C(u) = \{\langle y, z\rangle\}$ whose elements are sorted in y-and z-coordinates, resp.

begin

$Q := \emptyset$;

transform S into $A = t_1\$ \cdots \$t_m\$$; compute C and doc over A;

compute the suffix tree $Tree_A$ for A and suffix arrays suf, pos;

$INTER_k := \{\ \langle p, q\rangle \mid 1 \leq p,\ q \leq n,\ 0 \leq (q-p) \leq k,\ doc(p) = doc(q)\ \}$;

$R_k := \{\ \langle pos(p), pos(q), doc(p)\rangle \mid (p, q) \in INTER_k\ \}$;

Foreach node u in $Tree_A$ **do** /* traversing $Tree_A$ from the leaves to the root */

  if u is the x-th leaf l_x **then** initialize the list $B(l_x) := \{\ \langle y, z\rangle \mid \langle x, y, z\rangle \in R, \exists y, \exists z\ \}$;

  if u is an internal node with children $u_1, \ldots, u_h$ **then** update $B(u) := \cup_{1 \leq i \leq h} B(u_i)$;

   /* $B(u)$ is sorted in the x-coordinate without duplicates */

  Foreach node v in $Tree_A$ **do** /* traversing $Tree_A$ from the leaves to the root */

   if v is the y-th leaf l_y **then** initialize the list $C(l_y) := \{\ \langle z\rangle \mid \langle y, z\rangle \in B(u), \exists z\ \}$;

   if v is an internal node with children $v_1, \ldots, v_h$ **then**

    update $C(v) := \cup_{1 \leq i \leq h} C(v_i)$;

     /* $C(v)$ is sorted in the z-coordinate without duplicates */

   $P := (Word(u), k, Word(v))$;

   /* Now, $C(v)$ exactly contains all document numbers in which P occurs */

   compute $supp(P)$ and $conf(P)$ from the sorted list $C(v)$;

   if $supp(P) \geq \sigma$ **then**

    insert P into Q with the key $conf(P)$;

  end;

end;

Output all the patterns $P \in Q$ that has the highest confidence $conf(P)$;

end

Figure 2: A modified algorithm for discovering the optimized confidence patterns

Theorem 7 *Let S be a text database, C be an objective condition over C, $k \geq 0$ and $0 \leq \sigma \leq 1$ be fixed constants. Then, algorithm $Modified\_Find\_Optimal$ in Figure 2 computes all the optimized confidence patterns in canonical form with proximity k and support threshold σ in time $O(mn^2)$ and space $O(kn)$, where $m = card(S)$, $n = size(S)$. Furthermore, if the height of the suffix tree is d then it runs in time $O(kd^2n)$ and space $O(kn)$.*

Proof: We can see the results by easy calculations on Figure 2. At any stage of the computation, every element of R_k is contained by exactly one of $B(u)$'s and at most one of $C(v)$'s, and this gives the space $O(kn)$. The length of each list $B(u)$ or $C(u)$ is bounded by m, and this gives the time $O(mn^2)$ in m, n. Finally, each layer, the set of nodes at the same level, contains totally $N = kn$ elements of $B(u)$ (or $C(v)$), and This derives the time $O(kd^2n)$ in k, d, n. We omit the details. □

By the theorem, if the height of the suffix tree is $O(\log n)$ as in random texts or genetic sequences then the algorithm runs in time $O(kn \log^2 n)$.

5 Pruning and Sampling

Pruning: Based on the monotonicity of the support of patterns in canonical form $(W(u), k, W(v))$,

- If u is a parent of v then $supp_S(u) \geq supp_S(v)$,
- If $\min\{supp_S(W(u)), supp_S(W(v))\} < \sigma$ then $supp_S(\langle W(u), k, W(v) \rangle) \leq \sigma$,

we incorporate two pruning heuristics in the first algorithm: (1) Local pruning. Prune the descendants of u if $supp_S(W(u)) \leq \sigma$ at some u. (2) Global pruning. Prune the descendants of v if $supp_S(\langle W(u), k, W(v) \rangle) \leq \sigma$, where $supp(\alpha)$ is the support of a subword α. The local pruning is also possible in the second algorithm. By a similar argument to the proof of Theorem 7, we know that there are at most kd^2n canonical patterns of nonzero support for the height d of the suffix tree. Thus, we can expect that the efficiency of the first algorithm is improved with pruning for nearly random texts.

Sampling: The modified algorithm in Section 4 achieves $O(mn^2)$ time but it is not fast enough to be applied for huge text databases of several giga bytes. The following procedure approximates the solutions by using a random sampling technique.

Given: a sample S consisting of n examples.
  begin
    Draws m documents from S according to the uniform
    distributions. Let S_m be the obtained sample.
    By using algorithm *Find_Optimal*, compute the optimized
    confidence patterns P with respect to S_m, and output P.
  end

We set the sample size m to be, say, $O(n^{1/3})$ so that the algorithm works in almost linear time in n. The patterns computed by random sampling may give lower confidence than the patterns obtained from the original sample S. Therefore, we present empirical evaluations of the sampling heuristics by experiments.

6 Experimental Results

We run experiments on genetic data to evaluate the efficiency and the performance of our algorithms. The program was written in C based on the second algorithm in Section 4 and run on Sun Ultra 1 workstation under the Sun Solaris 2.5 operating system. The data were amino acid sequences of totally $24KB$ from GenBank database (1991). We obtained 450 positive sequences related to the signal peptide and 450 negative sequences 450 from other sequences, and preprocessed the data by transforming twenty amino acids into three