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

Hiroki Arimura, Atsushi Wataki*, Ryoichi Fujino, and Setsuo Arikawa

Department of Informatics, Kyushu University
Hakozaki 6-10-1, Fukuoka, 812-8581 Japan
{arim,wataki,fujino,arikawa}@i.kyushu-u.ac.jp
http://www.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-words 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 high probability. The optimized confidence pattern problem is to compute frequent patterns (α, k, β) that optimize the confidence with respect to a given collection of texts. Although this problem is solved in polynomial time by a straightforward algorithm that enumerates all the possible patterns in time $O(n^5)$, we focus on the development of more efficient algorithms that can be applied to large text databases. We present an algorithm that solves the optimized confidence pattern problem in time $O(\max\{k, m\}n^2)$ 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 $30 \sim 50$. This algorithm combines the suffix tree data structure in combinatorial string matching and the orthogonal range query technique in computational geometry for fast computation. Furthermore for most random texts like DNA sequences, we show that a modification of the algorithm runs very efficiently in time $O(kn \log^3 n)$ and space $O(kn)$. We also discuss some heuristics such as sampling and pruning as practical improvement. Then, we evaluate the efficiency and the performance of the algorithm with experiments on genetic sequences. A relationship with efficient Agnostic PAC-learning is also discussed.

1 Introduction

The recent progress of communication and network technologies, e.g., electronic mail, World Wide Web, and inter/intra networks makes it easy for computer users to accumulate a large collection of unstructured or semi-structured texts on their computers at a low cost [1]. Such *text databases* may be collections of

* Presently working for Fujitsu LTD.

web pages or SGML documents (OPENTEXT Index [26]), protein databases in molecular biology (GenBank [11]), online dictionary (OED [13]), 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 [1, 19, 28].

Data mining is a research area that aims at development of semi-automatic tools for discovering useful information from large databases. Data mining emerged in early 1990's and has been extensively studied in both practice and theory [2, 10, 15]. However, the present data mining technologies mainly deal with well-structured data such as relational databases with Boolean or numeric attributes [2, 10, 15], and thus are not directly applicable to those unstructured text data. Because, a text database is simply a collection of unstructured strings and the amount of the data is huge, which typically ranges from mega (10^6) bytes to tera (10^{12}) bytes. We concentrate on efficient and robust discovery methods that work for a large collection of unstructured texts [9, 19, 23, 28].

In this paper, we consider the discovery of very simple patterns called k-proximity two-words association patterns. Given a collection S of texts and an objective condition C over S, a *k-proximity two-words association 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 $k = 30$ letters then the objective condition C will hold with a probability. Although this class of rules seems very restricted, they are more flexible to describe a local similarity in text data with context information than unordered collections of keywords or single subwords. Hence, this kind of rules are frequently used for applications in bioinformatics [12], bibliographic search [13], and Web search [26]. Further, the simplicity of the class allows robust and efficient learning in noisy environment as we will show in this paper.

As the framework of data mining, we adopt the *optimal pattern discovery* recently proposed by Fukuda *et al.* [10]. In their framework, a discovery algorithm receives a sample set S with an objective condition $C : S \to \{0, 1\}$ and finds all/some patterns P that maximize a certain criterion. Based on this framework, we consider the efficient solvability of the *optimized confidence pattern problem* in data mining (Fukuda *et al.* [10]) and the *empirical error minimization problem* in computational learning theory (Kearns,Shapire, Sellie [17]; Maass [20]).

The optimized confidence patterns can be computed in time $O(n^5)$ by a straightforward algorithm that enumerates $O(n^4)$ possible two-words 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 and in Section 4, we present variations of algorithms that compute all the two-words 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 such patterns to that of axes-parallel rectangles over the 2-dimensional plane of suffix ranks.

In Section 3, we present an algorithm that 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 $O(\max\{k, m\}n^2)$ and space $O(kn)$ in the worst case. For most of nearly random texts like DNA sequences, it is known that the depth of the suffix tree is bounded by $O(\log n)$. Then, we show that our algorithm can be modified to run very quickly in time $O(kn \log^3 n)$ for such random texts.

In Section 5, we describe an application to computational learning theory. We show that the algorithm in Section 4 can be modified to efficiently solve the empirical error minimization problem. As a corollary, we obtain an efficient Agnostic PAC-learner for the class of k-proximity two-words association patterns. Nakanishi *et al.* [25] study the consistency problem, which is a special case of the empirical error minimization problem, for the class of single substrings. Since a single substring is a special k-proximity two-words association pattern, our result also generalizes their result.

In Section 6, we introduce some heuristics and examine their performances. In Section 7, we evaluate the efficiency and the performance of our algorithm from the empirical view point with experiments on genetic sequences from GenBank databases. Finally, Section 8 concludes the results.

2 Preliminaries

2.1 Texts and patterns

Let Σ be a finite alphabet. We always assume a fixed total order on the letters in Σ. 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. 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. A *text* is any string t over Σ. For $1 \leq i \leq |t|$, we denote the i-th letter of t by $t[i]$. An *occurrence* of $s \in \Sigma^*$ in t is a positive integer i such that $t[i] \cdots t[i + |t| - 1] = s$.

Let k be a nonnegative integer. A *k-proximity two-words association pattern* (or pattern, for short) is a triple $P = (\alpha, k, \beta)$, where $\alpha, \beta \in \Sigma^*$ are strings over Σ and $k \geq 0$ is called a *proximity*. A proximity two-words association pattern P *matches* a string $t \in \Sigma^*$ if there exist a pair of integers p and q such that p and q are the occurrences of α and β in t, respectively, and satisfy $0 \leq q - p \leq k$. The pair (p, q) is called an *occurrence* of P in t. 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.

2.2 Data mining problem

A *sample* is a finite set $S = \{s_1, \ldots, s_m\}$ of strings over Σ and an *objective condition over* S is a binary labeling function $C : S \to \{0, 1\}$. Each string $s_i \in S$ is a *document* and $C(s_i)$ is its *label*. A document s_i is *positive* if $C(s_i) = 1$ and *negative* otherwise. Let $\alpha \in \{0, 1\}$ be any label. We denote the set $\{ s \in S \,|\, C(s) = \alpha \}$ by S_α. For a subset T of S, we define $count(P, T)$ to be the number of documents $s \in T$ that P matches.

The first problem to consider is the confidence maximization [10]. For a pattern P, we define the *support* of P by $supp_S(P) = count(P, S_1)/card(S)$ and the *confidence* of P by $conf_S(P) = count(P, S_1)/count(P, S)$. A *minimum support* is any real number $0 \leq \sigma \leq 1$. A pattern P is said to be *σ-frequent* if $supp_S(P) \geq \sigma$.

Definition 1. Optimized Confidence Pattern Problem
An instance is a five-tuple $(\Sigma, S, C, k, \sigma)$ of an alphabet Σ, a sample set S, an objective condition C over S, constants $k \geq 0$ and $0 \leq \sigma \leq 1$. The problem is to find a σ-frequent k-proximity two-words association pattern P that maximizes $conf_S(P)$.

Intuitively, the optimized confidence pattern problem is to find an implication $P \Rightarrow C$ with highest conditional probability among those rules that can apply to at least σ percent of the documents in S. In Section 3 and Section 4, we give algorithms for efficiently solving the optimized confidence pattern problem.

The second problem to consider is the empirical error minimization [17, 20]. Let S be a sample, C an *objective condition over* S and P a k-proximity two-words association pattern over Σ. We define $P(s) \in \{0, 1\}$ is 1 iff P occurs in s. The *empirical error* of a pattern P with respect to S and C is the number of the documents in S misclassified by P, that is, $error_{S,C}(P) = \sum_{s \in S} |P(s) - C(s)|$.

Definition 2. Empirical Error Minimization Problem
An instance is a four-tuple (Σ, S, C, k) of an alphabet Σ, a sample set S, an objective condition C over S, a constant $k \geq 0$. The problem is to find a k-proximity two-words association pattern P that minimizes $error_{S,C}(P)$.

As we will see in Section 5, the empirical error minimization problem is closely related to a learning problem in noisy environments.

2.3 Suffix trees

A suffix tree is a data structure for storing all subwords of a given text in very economical way (McCreight [24]). 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 suffices 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, $A[p]A[p+1]\cdots[q] = \alpha$. (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 suffices of A.

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 $Word(w)$ is a proper prefix of α, where w is the parent of v.

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 a random string of length n is $O(\log n)$ [7]. This is also the case for genetic sequences.

2.4 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 a method described in Preparata and Shamos [27] 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 extend 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. The keys of the algorithm are steps to enumerate patterns in canonical form and to compute $supp(P)$ and $conf(P)$ quickly. Let $(\Sigma, S, C, k, \sigma)$ be an instance of the optimized confidence pattern problem.

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

```
  begin
1   Q := ∅;
2   A := s_1$ ··· $s_m$ and compute doc;
3   Build the suffix tree Tree_A and suffix arrays suf, pos.
4   Compute Diag_k and Rank_k from A.
5   Foreach node v do compute I(v);
6   Foreach node u in Tree_A do      /* Traversing Tree_A from the root to the leaves */
7     Foreach node v in Tree_A do      /* Traversing Tree_A from the root to the leaves */
8       P := (Word(u), k, Word(v));
9       Compute count(P, S_1) and count(P, S_0) by making
10      an orthogonal range query I(u) × I(v) for Rank_k;
11      Compute supp(P) and conf(P);
12      if supp(P) ≥ σ then insert P into the priority queue Q with the key conf(P);
13    end;
14  end;
15  Output all the optimized confidence patterns in Q;
  end
```

Fig. 1. An algorithm for discovering the optimized confidence patterns

3.1 Enumerating the patterns in canonical form using a suffix tree

Let $\$ \notin \Sigma$ be a symbol such that $\$ \neq \$$. Given $S = \{s_1, \ldots, s_m\}$ and $C : S \to \{0, 1\}$ as input, our algorithm computes a single text $A = s_1\$ \cdots \$s_m\$$ called an *input text* by concatenating all documents in S delimited with $\$$. Let $n = |A|$. For each $1 \leq p \leq n$, we define $doc(p) = i$ if if i-th text $s_i \in S$ includes p. Without loss of generality, we assume that there exists some $1 \leq p \leq m$ such that $C(s_i) = 1$ for all $1 \leq i \leq p$ and $C(s_i) = 0$ for all $p < i \leq m$.

Next, we build the suffix tree $Tree_A$ for input text A by using a linear time algorithm in [24]. This tree $Tree_A$ is isomorphic to the *generalized suffix tree* (GST), that is, the compacted trie for all the suffices of documents in S, except the labels of the edges directed to the leaves (Amir *et al.* [3]). Then, for each leaf v, we redefine $Word(v)$ as the longest prefix of the suffix represented by v that contains no $\$$'s. This is actually a standard method to build GST in linear time [3].

Now, we introduce an equivalence relation $\equiv_A$ as follows. For strings α, β, $\alpha \equiv_A \beta$ iff $Occ_A(\alpha) = Occ_A(\beta)$. For k-proximity two-words association patterns $P = (\alpha_1, k, \alpha_2), Q = (\beta_1, k, \beta_2)$, $P \equiv_A Q$ iff $\alpha_1 \equiv_A \beta_1$ and $\alpha_2 \equiv_A \beta_2$. If $P \equiv_A Q$ then we say that P and Q are *equivalent*.

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

Proof. By definition, equivalent patterns P, Q have the same set of the occurrences for any text A. Thus, $count(P,T) = count(Q,T)$ for any subset $T \subseteq S$. Since $supp_S(P)$ and $conf_S(P)$ are defined through $count$, 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 $Tree_A$. By definition, the number of k-proximity two-words association patterns is $O(n^2)$. Let $\perp_S \in \Sigma^*$ be an arbitrary string whose length is $\max\{\,|s| \mid s \in S\} + 1$. Clearly, $\Delta_{S,\xi}(\perp_S) = 0$.

Lemma 2. *For any k-proximity two-words association pattern P, if P matches a document in S then there exists a equivalent pattern in canonical form.*

Proof. Let $P = (\alpha, k, \beta)$. We show that for any subword α of A, there exists a node w of $Tree_A$ such that $\alpha \equiv_A Word(w)$ as follows. Suppose that we have the uncompacted suffix trie $\widehat{Tree}_A$ for A. Then, there exists a node v of $\widehat{Tree}_A$ that represents α. Let w be the highest descendant of v that has at least two children. Now, we map the nodes in $Tree_A$ into those in $\widehat{Tree}_A$ in a standard way. Then, we can easily see that v and w are mapped on a same edge in the compacted version $Tree_A$. We know that $subtree(v)$ and $subtree(w)$ have the same set of leaves, and thus they have the same set of occurrences in A. Since $w = Locus(\alpha)$ in $Tree_A$, we have $Word(w) \equiv_A \alpha$. Hence, the lemma follows. □

Lemma 3. *For any optimized confidence pattern among k-proximity two-words association patterns, there is an equivalent pattern P that is either in canonical form or $\perp_S$.*

3.2 Computing the support values using range queries

In this section, we show how to quickly compute the support and the confidence by using orthogonal range queries. The technique used here is basically due to Manber and Baeza-Yates [21]. The idea is to reduce the discovery of optimized confidence patterns to the discovery of 2-dimensional axes-parallel rectangles over the space consisting of the ranks of the lexicographically ordered suffices.

Let $A_{p_1}, A_{p_2}, \ldots, A_{p_n}$ be the sequence obtained by arranging all the suffices of A in the lexicographic order over Σ^*. Here, 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 $suf : [1..n] \to [1..n]$ 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* (Gonnet and Baeza-Yates [14]; Manber and Myers [22]). By definition, $suf(i)$ is the position of the suffix of rank i and $pos(p)$ is the rank of the suffix A_p.

First, we observe that for any node v of $Tree_A$, the set of the occurrences of $Word(v)$ occupy a contiguous subinterval $I(v) = [x_1..x_2]$ in array suf.

Lemma 4. *The subintervals $I(v)$ for all nodes v are computed in $O(n)$ time.*

Proof. $I(v)$ is computed by traversing $Tree_A$ from the leaves to the root in, e.g., depth-first search. Scanning the leaves from left to right, let $I(v_i) = [i..i]$ for the i-th leaf ($i = 1, \ldots, n$). Then, traversing $Tree_A$ from the leaves to the root, we visit each internal node v with its children $v_1, \ldots, v_h$ and let $I(v) = [L..R]$, where L and R are the left boundary of $I(v_1)$ and the right boundary of $I(v_h)$. □

Secondly, the algorithm builds from S and C the *diagonal set of width k*, that is, the set of 2-dimensional labeled points

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

Then, the algorithm transforms $Diag_k$ into the following set of 2-dimensional labeled points:

$$Rank_k = \{\ (pos(p), pos(q); i) \mid (p, q; i) \in Diag_k\ \}.$$

For a k-proximity two-words association pattern $P = (\alpha, k, \beta)$, we associate a 2-dimensional axis-parallel rectangle $Rect(P) = I(\alpha) \times I(\beta)$, and define $count(Rect(P), Q)$ to be the number of distinct i such that $(x, y; i) \in Q \cap Rect(P) \times [1..m]$, where Q is any subset of $Rank_k$.

Lemma 5. *For any k-proximity two-words association pattern P, $count(P, S) = count(Rect(P), Rank_k)$.*

From Lemma 5, the problems of computing $supp_S(P)$ and $conf_S(P)$ reduces to the problem of computing $count(Rect(P), Rank_k)$. From [27], a standard argument show the following lemma.

Lemma 6. *Let l be any integer, $Q \subseteq [1..l]^2 \times [1..m]$ be a set of 2-dimensional labeled points and $R \subseteq [1..n]^2$ be any rectangle. Then, $count(R, Q)$ is computable in time $O(m \log^2 n)$ and space $O(mn \log n)$ with preprocessing time $O(n \log n)$, where n is the number of points in Q.*

Theorem 1. *Let $(\Sigma, S, C, k, \sigma)$ be an instance of the optimized confidence pattern problem. Then, the 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 3, 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 5 and Lemma 6 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. Note that the number of $Rank_k$ is at most kn. 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 $O(\max\{k,m\}n^2)$ and space $O(kn)$, In the algorithm, we implement an orthogonal range query mechanism over the suffix tree itself instead of a range tree and compute the value of $count(Rect(P), Rank_k)$ by a dynamic programming in a similar way as Maass [20]. Figure 2 shows a modified version of our algorithm.

Procedure: *Modified_Find_Optimal*;
Given: a sample $S = \{s_1, \ldots, s_m\}$, an objective condition C, constants $k \geq 0$, and $0 \leq \sigma \leq 1$.
Output: the optimized confidence patterns (α, k, β) in canonical form.
Variable: a priority queue Q;
begin
1 Compute $A = s_1\$ \cdots \$s_m\$$, and doc; $Q := \emptyset$;
2 Build the suffix tree $Tree_A$ and suffix arrays suf, pos;
3 Compute $Diag_k$ and $Rank_k$ from S.
4 **Foreach** node u in $Tree_A$ traversing from the leaves to the root **do**
5 **if** u is the x-th leaf l_x **then**
6 $B(l_x) := \{ \langle y, z\rangle \mid \langle x, y, z\rangle \in Rank_k, \exists y, \exists z \}$;
7 **if** u is an internal node with children $u_1, \ldots, u_h$ **then**
8 $B(u) := B(u_1) \cup \cdots \cup B(u_h)$, and then discard $B(u_1), \ldots, B(u_h)$;
9 **Foreach** node v in $Tree_A$ traversing from the leaves to the root **do**
10 **if** v is the y-th leaf l_y **then**
11 $C(l_y) := \{ \langle z\rangle \mid \langle y, z\rangle \in B(u), \exists z \}$;
12 **if** v is an internal node with children $v_1, \ldots, v_h$ **then**
13 $C(v) := C(v_1) \cup \cdots \cup C(v_h)$ and then discard $C(v_1), \ldots, C(v_h)$;
14 $P := (Word(u), k, Word(v))$;
15 Compute $count(P, S_1)$ and $count(P, S_0)$ from $C(v)$;
16 **if** $supp(P) \geq \sigma$ **then**
17 Insert P into Q with the key $conf(P)$;
18 **end**;
19 **end**;
20 Output all the optimized confidence patterns in Q;
end

Fig. 2. A modified algorithm for discovering the optimized confidence patterns

Each node v of $Tree_A$ has linear list $B(v)$ and $C(v)$. Elements of $B(v)$ are pairs $\langle y, z\rangle \in [1..n] \times [1..m]$ and are sorted in the y-coordinate y. Similarly, Elements of $C(v)$ are labels $\langle z\rangle \in [1..m]$ and are sorted in the z-coordinate z. For any $x \in [1..n]$, we denote by l_x the x-th leaf.

Lemma 7. *Suppose the algorithm Modified_Find_Optimal visits node u in the loop at Line 4, visits node v in the loop at Line 9, and reaches Line 15. Then, $C(v)$ is the ordered list of the z's such that $(x, y; z) \in Rank_k \cap Rect(P) \times [1..m]$.*

Proof. Proved by induction on the height of the node u and v in a similar way as in Lemma 4. □

Theorem 2. *Let $(\Sigma, S, C, k, \sigma)$ be an instance of the optimized confidence pattern problem. Then, the 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 $O(\max\{k,m\}n^2)$ and space $O(kn)$, where $m = card(S)$, $n = size(S)$.*

Proof. The correctness follows from Lemma 1, Lemma 3, Lemma 5, and Lemma 7. Then, we estimate the running time. Line 1 to Line 3 take time $O(n+kn)$. Since $Tree_A$ contains $O(n)$ nodes, the outer loop at Line 4 is executed $O(n)$ times, and thus the inner loop at Line 9 is executed $O(n^2)$ times. At each visit to a node u in the outer loop, each merge operation at Line 8 takes time at most $O(kn)$ because the length of the resulting list $B(u)$ is obviously bounded by $kn = card(Rank_k)$. Line 6 takes total time $O(kn)$ in scanning all leaves. Similarly, at each visit to a node v in the inner loop, each merge operation at Line 13 takes time $O(hm)$ because each $C(v_i)$ are sorted and its length is bounded by m, where h is the number of children. Since the sum of the number of the children at all nodes is $O(n)$, amortized total time of Line 13 is $O(mn)$ and that of Line 11 is $O(kn)$ in scanning all leaves for a fixed u. Combining these observations, the total running time of the algorithm is $T(n) = O(kn)+O(kn)+n\cdot(O(kn)+O(mn)) = O(\max\{k,m\}n^2)$. □

It is known that the height of the suffix tree is $O(\log n)$ for random texts generated by a uniform distribution [7]. By Theorem 3 below, we expect an algorithm to run very efficiently in time $O(kn\log^3 n)$ with a high probability for such random texts.

Theorem 3. *Let $(\Sigma, S, C, k, \sigma)$ be an instance of the optimized confidence pattern problem. Then, there exists an algorithm that computes all the optimized confidence patterns in canonical form with proximity k and support threshold σ in time $O(kd^2 n\log n)$ and space $O(kn)$, where $n = size(S)$ and d is the height of the suffix tree.*

Proof. We observe that each layer, the set of nodes at the same level, of a suffix tree contains totally $N = kn$ elements in $B(u)$ $(C(v))$. In the modified algorithm, we attach with each node the pointer to its parent and a sorted list $C(v)$ and $B(u)$ of points $\langle y, z\rangle$ and $\langle x, y, z\rangle$. We use a balanced tree to represent each list $C(v)$ so that we can perform both the insertion of an element and the counting of positive-labeled or negative-labeled elements in time $\log l$ for $l = card(C(v))$. By using these, we traverse the tree levelwise from the leaves to the root. Thus, at each node u with list $B(u)$, we can compute all $C(v)$ in time $O(dn_0\log n_0)$ from Line 10 to Line 17 in Figure 2, where $n_0 = card(B(u))$. Repeatedly using this idea, we can derive the time $O(kd^2 n\log n)$ in k, d, n. We omit the details. □

5 Agnostic PAC-learning and the empirical error minimization

Agnostic PAC-learning is a generalization of a well known PAC-learning model in computational learning theory, which is intended to capture learning situations in real world [16, 17, 20]. In agnostic PAC-learning, a learning algorithm must work robustly with noisy environments, and even have the capability to *approximate* arbitrary unknown probability distributions.

Haussler [16] and Kearns *et al.* [17] showed a close link between Agnostic PAC-learning and the empirical error minimization problem defined above. The *Vapnik-Chervonenkis dimension* (VC-dimension) is a measure of the complexity of a concept class (See, e.g. [16, 17] for the definition). The class of k-proximity two-words association patterns obviously has polynomial VC-dimension for any fixed $k \geq 0$.

Lemma 8 (Kearns *et al.* [17]). *For any hypothesis class with polynomial VC-dimension, the polynomial time solvability of the empirical error minimization problem and the efficient agnostic PAC-learnability are equivalent.*

Now, we show that we can modify the algorithm *Modified_Find_Optimal* to solve the empirical error minimization problem in the same time and space complexity as the original version.

Theorem 4. *Let S be a sample, C be an objective condition over S, $k \geq 0$ be a fixed constant. Then, there exists an algorithm that solves the empirical error minimization problem for k-proximity two-words association patterns in time $O(kd^2 n \log n)$ and space $O(kn)$, where $n = size(S)$ and d is the height of the suffix tree.*

Proof. We can easily show that the empirical error minimization is obviously equivalent to the maximization of the difference $\Delta_{S,C}(P) = count(P, S_1) - count(P, S_0)$. It is not hard to see that $\Delta_{S,C}$ is maximized by either a pattern in canonical form or $\perp_S$ as in Lemma 3. Now, we modify the algorithm *Modified_Find_Optimal* in Figure 2 to compute a pattern P that maximizes $\Delta_{S,C}(P)$. At Line 15 of the algorithm, we compute the quantities $count(P, S_1)$ and $count(P, S_0)$, and then compute $\Delta_{S,C}(P)$. We skip the test at Line 16, and then at Line 17 we insert P into a priority queue Q with $\Delta_{S,C}(P)$ as the key. This modification correctly works, and hence the theorem follows from Theorem 3. □

Although the empirical error minimization problem is intractable for most concept classes, recently some geometrical patterns, e.g. *axis-parallel rectangles* and *convex k-gons* on Euclidean plain, are shown to be efficiently agnostic PAC-learnable [8, 20]. From Lemma 8 and Theorem 4, we know that there exists an agnostic learning algorithm that runs in time $O(kn \log^3 n)$ if an input sample is ensured to be random. Hence, our result is one of a few results on the efficient agnostic learnability for nontrivial concept classes other than geometric patterns.

6 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 2, 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.

7 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 [11]. 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 symbols $0, 1, 2$ by indices due to Kyte and Doolittle (1982). For each $m = 10 \sim 100$ and each trial, we randomly drawn m positive and m negative sequences from the original sample S, and compute the optimum patterns for the obtained sample.

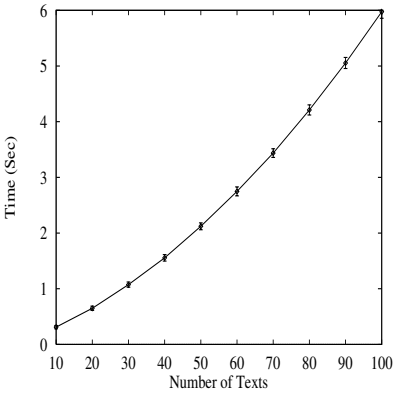

Fig. 3: The running time

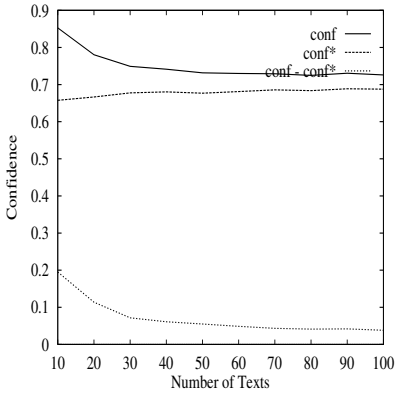

Fig. 4: The performance of sampling

Running time. Figure 3 shows the dependency of the running time to the number m of documents, which is proportional to the sample size n. We see that the growth of the running time is slower than that we expected from Theorem 2. For a larger sample of 24KB ($m = 450$), the algorithm takes one minute and over twenty mega bytes of main memory to find 37 optimal patterns with testing 676 locally frequent patterns within 577296729 possible patterns. Examples of the optimal patterns are "12222"*"2" and "0"*"2222" that achieve high confidence 69% and 66% with support around 65%.

Random sampling. Figure 4 show the performance of sampling heuristics for varying sample size. For each trial, first compute the best ten patterns for the sample S_m and then evaluate the empirical confidence $conf$ on S_m and the real confidence $cond^*$ on S. We plotted the average of the confidence through 100 trials for each m. The parameters were $k = 5$ and $\sigma = 0.6$. In Figure 4, we can see that the error is 5% for the confidence by 10% sampling. For the support, we had a similar result (12% error for the support).

Table 1. The performance of pruning heuristics

σ %	Sample size(bytes)	Candidates	Locally frequent	Globally frequent
90%	2,701	7,295,401	121	40
50%	2,754	7,584,516	1,444	64
30%	2,764	7,639,696	6,241	88

Pruning. Table 1 shows the efficiency of the pruning heuristics. The sample consists of 50 positive and 50 negative sequences of totally 2.7KB. The first and the second columns show the minimum support and the sample size, and the third, the fourth, and the fifth columns show the number of candidate patterns remain at the initial, after local pruning, and after global pruning. We can see that only a small fraction of patterns can be solutions for nearly random sequences.

8 Conclusion

In this paper, we considered the problem of finding association rules over subwords, and gave efficient algorithms that compute optimized confidence rules from a large collection of unstructured text data for the class of two-words association patterns. We discussed the performance of search heuristics and run experiments on genetic data.

The algorithms presented in this paper extensively use the *suffix array* data structure, which is a variant of the suffix tree that is more space efficient and suitable for implementing advanced search functions in large text databases (Gonnet and Baeza-Yates [14]; Manber and Myers [22]). Thus, it is a future problem to develop scalable implementation techniques over suffix arrays that enables us to incorporate our algorithms into existing text databases [13, 26]. The study of secondary storage directed algorithms and the speed-up by parallel execution are other future problems.

We developed efficient learning algorithm for classes of structured patterns [4–6]. It will be interesting to extend the framework of this paper for the structured patterns.

Acknowledgments

The authors would like to thank anonymous referees for their suggestions that greatly improve the correctness and the presentation of this paper. We also thank Shinichi Shimozono, Masayuki Takeda, Ayumi Shinohara, Takeshi Shinohara and Thomas Zeugmann for their valuable discussions and comments concerning this work. The first author would like to thank Esko Ukkonen and Heikki Mannila for directing his interests in this area. This research is partly supported from the Ministry of Education, Science and Culture, Japan by Grant-in-Aid for Scientific Research on Priority Area "Advanced Databases" and by Grant-in-Aid for Scientific Research on Priority Area "Discovery Science."

References

1. S. Abiteboul, Querying semi-structured data. In Proc. ICDT'97 (1997).
2. R. Agrawal, T. Imielinski, A. Swami, Mining association rules between sets of items in large databases. In Proc. the 1993 ACM SIGMOD Conference on Management of Data, 207–216 (1993).
3. A. Amir, M. Farach, Z. Galil, R. Giancarlo, K. Park, Dynamic Dictionary Matching. *JCSS*, 49 (1994), 208–222.
4. H. Arimura, R. Fujino, T. Shinohara, and S. Arikawa. Protein motif discovery from positive examples by minimal multiple generalization over regular patterns *Proc. Genome Informatics Workshop 1994*, 39–48, 1994.
5. H. Arimura, H. Ishizaka, T. Shinohara, Learning unions of tree patterns using queries, *Theoretical Computer Science*, 185 (1997) 47–62.
6. H. Arimura, T. Shinohara, S. Otsuki. Finding minimal generalizations for unions of pattern languages and its application to inductive inference from positive data. In Proc. the 11th STACS, LNCS 775, (1994) 649–660.

7. L. Devroye, W. Szpankowski, B. Rais, A note on the height of the suffix trees. *SIAM J. Comput.*, 21, 48–53 (1992).
8. D. P. Dobkin and D. Gunopulos, Concept learning with geometric hypothesis. In Proc. COLT95 (1995) 329–336.
9. R. Feldman and I. Dagan, Knowledge Discovery in Textual Databases (KDT). In Proc. KDD-95 (1995).
10. T. Fukuda, Y. Morimoto, S. Morishita, and T. Tokuyama, Data mining using two-dimensional optimized association rules. In Proc. the 1996 ACM SIGMOD Conference on Management of Data, (1996) 13–23.
11. GenBank, GenBank Release Notes, IntelliGenetics Inc. (1991).
12. G. Gras and J. Nicolas, FOREST, a browser for huge DNA sequences. In Proc. the 7th Workshop on Genome Informatics (1996).
13. G. Gonnet, PAT 3.1: An efficient text searching system, User's manual. UW Center for the New OED, University of Waterloo (1987).
14. G. Gonnet, R. Baeza-Yates, Handbook of Algorithms and Data Structures, Addison-Wesley (1991).
15. J. Han, Y. Cai, N. Cercone, Knowledge discovery in databases: An attribute-oriented approach. In Proc. the 18th VLDB Conference, 547–559 (1992).
16. D. Haussler, Decision theoretic generalization of the PAC model for neural net and other learning applications. Information and Computation 100 (1992) 78–150.
17. M. J. Kearns, R. E. Shapire, L. M. Sellie, Toward efficient agnostic learning. *Machine Learning*, 17(2–3), 115–141, 1994.
18. J. Kyte and R. F. Doolittle, In J. Mo. Biol., 157 (1982), 105–132.
19. D. D. Lewis, Challenges in machine learning for text classification. In Proc. 9th Computational Learning Theory (1996), pp. 1.
20. W. Maass, Efficient agnostic PAC-learning with simple hypothesis, In Proc. COLT94 (1994), 67–75.
21. U. Manber and R. Baeza-Yates, An algorithm for string matching with a sequence of don't cares. IPL 37, 133–136 (1991).
22. U. Manber and E. Myers, "Suffix arrays": a new method for on-line string searches. In Proc. the 1st ACM-SIAM Symposium on Discrete Algorithms (1990)319–327.
23. H. Mannila, H. Toivonen, Discovering generalized episodes using minimal occurrences. In Proc. KDD'96 (1996) 146–151.
24. E. M. McCreight, A space-echonomical suffix tree constructiooon algorithm. JACM 23 (1976), 262–272.
25. M. Nakanishi, M. Hashidume, M. Ito, A. Hashimoto, A linear-time algorithm for computing characteristic strings. In Proc. the 5th International Symposium on Algorithms and Computation (1994), 315-23.
26. OPENTEXT Index.
http://index.opentext.net (1997).
27. F. P. Preparata, M. I. Shamos, Computational Geometry. Springer-Verlag (1985).
28. J. T.-L. Wang, G.-W. Chirn, T. G. Marr, B. Shapiro, D. Shasha, K. Zhang. Combinatorial Pattern Discovery for Scientific Data: Some preliminary results. In Proc. 1994 SIGMOD (1994) 115–125.