Mining Peculiar Compositions of Frequent Substrings from Sparse Text Data Using Background Texts

Daisuke Ikeda and Einoshin Suzuki

Department of Informatics, Kyushu University
Motooka 744, Fukuoka 819-0395, Japan
{daisuke,suzuki}@inf.kyushu-u.ac.jp

Abstract. We consider mining unusual patterns from text T. Unlike existing methods which assume probabilistic models and use simple estimation methods, we employ a set B of *background* text in addition to T and *compositions* $w = xy$ of x and y as patterns. A string w is *peculiar* if there exist x and y such that $w = xy$, each of x and y is more frequent in B than in T, and conversely $w = xy$ is more frequent in T. The frequency of xy in T is very small since x and y are infrequent in T, but xy is relatively abundant in T compared to xy in B. Despite these complex conditions for peculiar compositions, we develop a fast algorithm to find peculiar compositions using the suffix tree. Experiments using DNA sequences show scalability of our algorithm due to our pruning techniques and the superiority of the concept of the peculiar composition.

1 Introduction

As text data, such as Web documents and genome sequences, are becoming abundant in various areas, it is becoming more important to analyze text data and to extract useful knowledge from it. Hence, an increasing attention has been paid to text mining.

Exceptionality has attracted attention of researchers in various areas as an essential property of discovered knowledge, since a discovery is often inspired by unusual events which can not be explained by the current theory. In the field of data mining, the term "unexpected" has been included as a mandatory property of the target patterns [1] from the beginning of the field and various methods have been proposed for finding exceptions, such as outliers [2], rules [3], and other types of patterns [4]. Unusualness for time series data has been also studied [5,6]. For text data, finding unusual text patterns have been studied extensively, especially in bioinformatics [7,8,9,10].

For the definition of being *unusual*, it is natural to define *usual* states and to measure unusualness by deviation from the states. In the z-score, which is a popular measure for unusual patterns in text data [8,9,10], the popularity of a usual state for a pattern w is an estimated expectation for w based on a given probability model. More formally, the z-score $z(w)$ for w is defined

$$z(w) = \frac{f(w) - E(w)}{N(w)},$$

where $f(w) > 0$ denotes the observed frequency, $E(w)$ its expectation and $N(w)$ a normalization factor. Given a threshold α, w is unusual if $z(w) > \alpha$ or $z(w) < -\alpha$. In other

W. Buntine et al. (Eds.): ECML PKDD 2009, Part I, LNAI 5781, pp. 596–611, 2009.

words, an unusual pattern based on the z-score is a pattern whose frequency deviates far from its estimate under the given probability model.

However, scores to measure unusualness borrowed from statistical testing, such as the z-score, have the following problems: (1) they require an appropriate probabilistic model in advance, (2) discovered unusual patterns lack of clear interpretations and (3) an estimation based on the probabilistic model is inappropriate for sparse text data.

Firstly, such a score requires an appropriate probabilistic model. A simple model is easy to compute the corresponding score but it can not describe details of given data, while a sophisticated model well describes the given data but it is computationally costly to obtain parameters for the model. When we assume the Bernoulli model, which is a simple model, we obtain the ζ-score as a variant of the z-score [7,8], where the expectation $\hat{p}(w)$ for a pattern w is given by $\hat{p}(w) = \prod p(a)$, which is the product of all probabilities $p(a)$ for letters a in w. In this model, each letter is supposed to occur independently. The Markov model is also considered for the probabilistic model of the z-score [10]: the probability of the ith letter a_i depends on the probability of the previous k letters and therefore $\hat{p}(w)$ is estimated by a conditional probability $p(a_i|a_{i-1} \cdots a_{i-k+1})$. However, the value of k is fixed and must be given in advance, and it is difficult to decide an appropriate value for k automatically.

Secondly, it is difficult for us to understand the meaning of patterns obtained by a score based on statistical testing. Such a pattern is evaluated from only the view point of the statistics. Therefore, we only know that the frequency of the pattern is rare or abundant against its expectation but the frequency does not tell us about its meaning.

Finally, the estimation based on simple probabilistic models is inappropriate to find unusual patterns from sparse text data because the models use short sub-patterns to estimate longer patterns' probabilities. Estimating probabilities for long patterns is critical to find unusual patterns since unusual patterns must be long from the definition of being unusual. However, the simple probabilistic models provide inaccurate estimates to all long patterns as many long patterns do not appear in practical sparse data. Thus, we need a better estimation method for sparse text data.

To overcome these problems, first, we introduce a *background set* in addition to a target set of text data. From the background set, we find sub-patterns which compose unusual patterns. In this sense, the background set is used as a mother population. In practical usage, it is natural to compare a given data with other data, instead of comparing with the population. For example, to examine DNA sequences of a species, the same or similar subsequences of other, well-known species can be a good hint.

Next, we introduce a new form of a pattern, called a *composition*, which is defined as the concatenation of two frequent substrings x and y. The lengths of x and y are not restricted and hence they can be long. Thus, we can expect x and y help us to understand implications of xy because x and y are enough long and frequent. For example, three nucleotides in genome sequences compose a codon and thus we might try to understand a sequence of nucleotides using the codon table in a bottom up manner.

Finally, we define a composition to be *peculiar* using two ratios of frequencies for both target and background sets. That is, $w = xy$ is peculiar if both x and y are more frequent in B than in T and conversely their composition xy is more frequent in T than in B. Therefore, the frequency of w is small in the target set since both x and y are

infrequent in T, and the frequency of w is larger than its expectation of $f(x)f(y)$ in B. In this sense, peculiar compositions are unusual. Hence, we only need to provide simple parameters to a data mining algorithm instead of a complicated probabilistic model.

Our problem definition raises a new challenging problem: we have to find frequent components as well as infrequent unusual patterns. Therefore, it is computationally hard to find peculiar compositions, compared to just finding substrings. We develop an algorithm, *FPCS* (Finding Peculiar Compositions), which exploits the generalized suffix tree and find peculiar compositions in $O(N^2)$ time, where N is the total length of input strings. The generalized suffix tree enables FPCS to find simultaneous discovery of a peculiar composition xy, its prefix x, and suffix y.

This paper is organized as follows: related work is surveyed in section 2. We define the problem in section 3 and then propose our main algorithm FPCS in section 4. Section 5 is devoted for experimental evaluation. Section 6 concludes.

2 Related Work

The z-score has attracted attention of bioinformatics researchers as a measure to find unusual patterns in strings. For a threshold α, if $z(w) > \alpha$ (resp. $z(w) < -\alpha$) then w is said to be *overrepresented* (resp. *underrepresented*). In [7,8], the Bernoulli model is assumed as a probabilistic model in estimating the expectations and the Markov model or the so called n-gram model is considered in [10]. The χ^2-score is also used to measure interestingness [9].

Many supervised learning algorithms, e.g., those for formal languages, and text mining algorithms [11,12,13,14] also use another set of text data in addition to a target set. However, they discover a pattern which satisfies many examples in a set but few in the other set. Therefore, patterns output by them are usual.

Indexing scores in information retrieval also consider a background set. For example, TF/IDF marks a high score to a word which appears frequently in a document but not in the background set. This criteria is similar to our notion of being peculiar. However, indexing scores are not for compositions of words but for single words, and the target set for TF/IDF is a single document not a set of documents like in our setting.

The concept of the peculiar composition borrows its essential idea from the simultaneous discovery of exceptional rules [15,16,17,18]. It tries to discover a set of rule pairs each of which corresponds to $Y \rightarrow x = v$ and $YZ \rightarrow x = v'$, where each of Y and Z represents a conjunction of "attribute = value"s, x is an attribute, v and v' are different values, and YZ is the conjunction of Y and Z. [15,16,18] assume that $Z \rightarrow x = v'$ does not hold and in this case a rule pair corresponds to a situation that an ensemble of two conditions result in an atypical result as shown in Fig. 1. Discovered rule pairs are shown to be valid, useful, novel, and unexpected in a medical application [18]. Our work can be considered as a

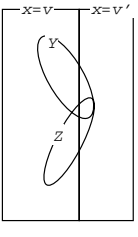

Fig. 1. Venn diagram of a discovered pattern in the simultaneous discovery of exception rules

text mining version of the simultaneous discovery of exception rules. It is difficult for algorithms in [15,16,17,18] to discover long patterns because the algorithms generate candidate patterns and then check their frequencies. One of our contributions lies in FPCS which exploits the generalized suffix tree in a sophisticated manner and hence the time complexity of FPCS does not depend on the lengths of x and y in discovered patterns.

3 Peculiar Composition Discovery Problem

We first define mandatory notions for dealing string data, introduce the peculiar composition, and its discovery problem.

Let Σ be a finite set of *characters*. We call Σ an *alphabet*. The set of all the finite sequences of zero or more characters is denoted by Σ^*. An element of Σ^* is called a *string*. We denote the length of a string x by $|x|$. The empty string, which is a string of zero character, is denoted by ε. The set of sequences each of which is composed of at least one character is denoted by Σ^+ i.e. $\Sigma^+ = \Sigma^* - \{\varepsilon\}$.

For strings $x, y \in \Sigma^+$, the concatenation of x and y is denoted by xy. We call xy the *composition* of x and y. For instance, if x = "every" and y = "thing" then xy = "everything". Conversely, a pair of two strings (x, y) is called a *division* of w if $w = xy$.

For a string $x = a_1 \cdots a_n$ ($a_i \in \Sigma$), if there exist $u, v, w \in \Sigma^*$ such that $x = uvw$ then u (resp. v and w) is a *prefix* (resp. *substring* and *suffix*) of x.[1] For instance, if uvw = "amazingly" then u="amaz" is a prefix, v="ing" is a substring, and w="ly" is a suffix. The notion of suffix will be crucial in introducing data structures in section 4.2. An *occurrence* of v in x is a positive integer i such that $a_i \cdots a_{i+|v|-1} = v$. The occurrence of "ing" in "amazingly" is 5, and "a" has two occurrences: 1 and 3. The *frequency* of v in x is the number of occurrences of v in x.

For $x, y \in \Sigma^*$, we consider that the frequency of x in y represents a kind of popularity of x in y. To treat this notion for a set D of strings, we use $\mathrm{f}(x|D)$ to denote the add-sum of the frequencies of x in all strings in D. Since the frequency is affected by the absolute size for D, we use relative frequencies or empirical probabilities $\mathrm{P}(x|D) = \mathrm{f}(x|D)/\#_D$, where $\#_D$ is the add-sum of frequencies of all substrings in D. For examplle, since AG appears 4 times in $D = \{CTAGAG, CTAGCTAG\}$ and $\#_D = 6 \cdot 7/2 + 8 \cdot 9/2 = 57$, $\mathrm{f}(AG|D) = 4$ and $\mathrm{P}(AG|D) = 4/57$.

We assume two sets T and B of strings, and we call T the *target* set and B the *background* set. Given a threshold $\theta > 1$, x is *contrastive* w.r.t. θ in target (resp. background) if $\mathrm{P}(x|T) > \theta\mathrm{P}(x|B)$ (resp. $\mathrm{P}(x|B) > \theta\mathrm{P}(x|T)$). θ represents the minimum ratio of the probability of x in one document set to that in the other set.[2] If it is clear from the context, we omit the threshold and simply say that x is contrastive in target or background. For instance, if T and B represent the sets of e-mails of A and B, respectively, and only A lives in Kyushu then "Kyushu" is likely to be contrastive w.r.t a relatively large θ.

Let $x, y \in \Sigma^+$. Given θ_T and θ_B, a composition xy is said to be *peculiar* in T against B if xy is contrastive in target w.r.t θ_T, and both x and y are contrastive in background

[1] A prefix or suffix is a substring because u or w can be the empty string.

[2] An appropriate value of θ in practice is decided by a trial and error process.

w.r.t θ_B. If it is clear from the context, we omit the string sets and simply call xy a peculiar composition.

In addition to the notion of being contrastive defined by the proportion of the relative frequencies, we also use the minimum support which is a minimum threshold for frequencies. For a set D of strings, we denote the minimum support by η_D, and we say that a string x is η_D-frequent in D if x appears more than η_D times in D. η_D represents the minimum support for the frequency of peculiar compositions.

If "Kyushu University" is a peculiar composition then its prefixes, such as "Kyushu Universi", are likely to be peculiar. These prefixes increase the number of patterns discovered by mining algorithms, but can be considered as irrelevant. Several pattern discovery algorithms discover only closed patterns which are maximal among equivalent patterns [19,20]. We define that two composition xy and $x'y'$ ($|xy| > |x'y'|$) are *equivalent* if $x = x'$, y' is a prefix of y, and $\mathrm{f}(y|D) = \mathrm{f}(y'|D)$ ($D = T, B$). We say that xy is *maximal* of equivalent compositions if xy is longest in the equivalent compositions.

The peculiar composition discovery problem is formally defined as follows:

Definition 1. *The peculiar composition discovery problem is, given two sets T and B of strings and threshold values θ_T, θ_B and η_T, to find all maximal, η_T-frequent, peculiar compositions in T against B.*

Example 1. Let $B = \{CTAGAG, CTAGCTAG\}$ and $T = \{AGCT, AGCT, AAAAAAA\}$. While AG and CT are popular but $AGCT$ is rare in B, $AGCT$ is popular in T. Thus $AGCT$ is a peculiar composition of AG and CT for $\theta_T = 1.2$, $\theta_B = 2.3$ and $\eta_T = 2$ because $\#_T = 48$, $\#_B = 57$, $\mathrm{f}(AGCT|T) = 2$ and $\mathrm{f}(AG|B) = \mathrm{f}(CT|B) = 3$.

It is easy to solve the problem if we neglect time-efficiency or if we limit the maximal length of the strings in the discovered patterns to a trivially-small number. A straightforward solution would be to count substrings and check if x, y and xy are contrastive. The number of possible substrings is $O(N^2)$, where N is the total length of input strings. For a substring of the form xy, we have to check $O(|xy|)$ compositions since there exist $O(|xy|)$ divisions. For each string w (or composition), $O(|w|)$ time is required to check if a string is contrastive. Thus the time complexity of this naive algorithm is $O(N^2 \times N \times N) = O(N^4)$. Certainly this solution is prohibitive for real problems.

The suffix tree [21,22], which is a popular data structure for text data, reduces this time complexity. Although there exist $O(N^2)$ possible substrings, we only need to check $O(N)$ substrings which correspond to nodes in a suffix tree, and it is done in constant time to check if a string is contrastive by storing frequencies on nodes of the tree. Thus, the time complexity is reduced to $O(N^2)$.

However, this is larger than the time complexity to compute z-score, which is calculated in $O(N)$ time [7]. The main difficulty arises from the fact that being contrastive does not satisfy anti-monotonicity because our problem is defined over two string sets instead of a single set. For a string w, there exist $O(|w|)$ divisions (x_i, y_i), where $w = x_i y_i$. Even if x_i, y_i and $x_i y_i$ are contrastive for some division (x_i, y_i), we do not conclude that, for another division (x'_i, y'_i), x'_i, y'_i and $x'_i y'_i$ are contrastive. Thus we need to check all divisions.

4 Our Algorithm FPCS

In this section, we present our main algorithm *FPCS* (Finding Peculiar Compositions).

4.1 Overview of FPCS

FPCS exploits the generalized suffix tree, which is a well-known data structure for text data. First FPCS constructs the generalized suffix tree for all input strings. A node of the tree corresponds to a substring, called a *branching string*, of the given strings. Then FPCS counts the frequencies for branching strings in both background and target sets simultaneously. As the result, the algorithm finds all nodes such that their branching strings are contrastive in target and, for such a node, there exists an ancestor node whose branching string is contrastive in background. Such a branching string xy is a candidate for a peculiar composition because xy is contrastive in target and x is contrastive in background. Finally FPCS checks if y is contrastive in background for all possible divisions (x, y) of xy efficiently using suffix links, which are pointers to nodes of the generalized suffix tree.

4.2 Generalized Suffix Tree

In this section, we briefly explain the generalized suffix tree used in FPCS. The generalized suffix tree is a suffix tree for a set of strings. The tree can be used to count all the frequencies of substrings in $O(N)$, where N is the total length of input strings. First we explain the suffix tree [21,22] for a single string and then extend it to the generalized suffix tree [23].

Let \$ be a special character such that $\$ \notin \Sigma$. For a string $x \in \Sigma^*$, $A = x\$$ and an integer p $(1 \leq p \leq |A|)$, A_p denotes A's suffix starting at the pth character of A. The special character \$ is used to guarantee any suffix of x is not a prefix of another suffix, and hence A_p is uniquely identified.

A trie for strings is the tree in which there is one node for every common prefix. A suffix trie for x is the trie for $A_{p_1}, A_{p_2}, \ldots, A_{p_n}$, where $A_{p_1}, A_{p_2}, \ldots, A_{p_n}$ are all suffixes of A in lexicographic order. The *suffix tree* for x is the compact trie for $A_{p_1}, A_{p_2}, \ldots, A_{p_n}$, where all nodes with one child are merged with their parents. The number of nodes in a suffix tree is $O(N)$, since the number of internal nodes is $O(N)$ due to the compactness of the tree and the number of leaves is exactly N.

For each node v of the tree, $BS(v)$ denotes the string obtained by concatenating all strings labeled on the edges on the path from the root to v. We call $BS(v)$ a *branching string*.

Example 2. Fig. 2 is the suffix tree for "*mississippi*\$". For nodes u and v, we have $BS(u) = issi$ and $BS(v) = i$.

A suffix tree is often used with suffix links to speed up related procedures. A suffix link is a pointer from a node u to another node w, where $|BS(w)| + 1 = |BS(u)|$ and $BS(w)$ is a suffix of $BS(u)$. In other words, $BS(w)$ is obtained by deleting the first character of $BS(u)$. Suffix links are generated simultaneously during the construction of the suffix

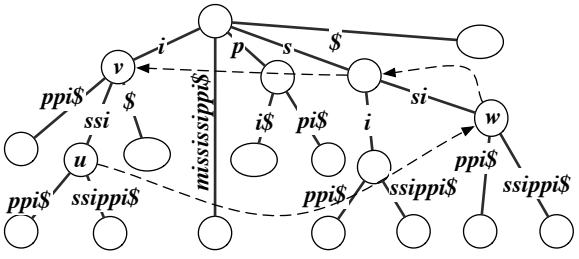

Fig. 2. The suffix tree for "*mississippi*$". A dotted line denotes a suffix link.

tree. In the above example, we have a suffix link from u to w because $BS(u) = issi$ and $BS(w) = ssi$.

For a node of the suffix tree of a string x, its branching string appears in x as many times as the number of leaves below the node. Node v of Fig. 2 has four leaves below, and so we find that $BS(v) = i$ appears four times in *mississippi*.

Let u be a child node of v. Then $BS(v)$ is a proper prefix of $BS(u)$ from the definition of the branching string. In Fig. 2, $BS(v) = i$ is a proper prefix of $BS(u) = issi$. Moreover, for a prefix s of $BS(u)$ which includes $BS(v)$ as a prefix, the frequency of s is the same as that of $BS(u)$. For example, $BS(u) = issi$ appears twice, and so do two of its prefixes *is* and *iss*. Therefore, when we count substring frequencies, all we have to do is to count only branching strings, i.e., to count leaves bellow nodes. Thus counting substring frequencies is completed in $O(N)$ time.

Now we introduce the generalized suffix tree to deal with a set D of strings instead of a single string. Let $D = \{x_1, x_2, \ldots, x_m\}$. The generalized suffix tree is the suffix tree for a string $x_1\$_1x_2\$_2 \ldots x_m\$_m$.

For a generalized suffix tree, we assume that a node v of the tree stores a pair of the frequencies for $BS(v)$ in background and target sets. Formally, this is defined recursively as follows: a leaf node v has $(1, 0)$ (resp. $(0, 1)$) if $BS(v) \in B$ (resp. T), and an internal node v has

$$\left(\sum_{u \in C(v)} \mathrm{f}(BS(u)|B), \sum_{u \in C(v)} \mathrm{f}(BS(u)|T) \right),$$

where $C(v)$ is the set of the children of v. Note that $BS(v)$ appears only once in the input strings for a leaf node v due to \$.

4.3 Details of FPCS

Now we explain the details of FPCS using Algorithm 1 and Algorithm 2. Algorithm 1 is the main procedure which finds candidates for peculiar compositions, and Algorithm 2, which is called from the main procedure, checks the conditions to be peculiar for each of the candidates.

Given two string sets, first FPCS constructs the generalized suffix tree and stores a pair $(\mathrm{f}(BS(v)|T), \mathrm{f}(BS(v)|B))$ of the frequencies for $BS(v)$ in background and target sets on a node during a postorder traversal (see Algorithm 1).

Algorithm 1. FPCS for discovery of all maximal, frequent peculiar compositions.

Input: T, B, θ_T, θ_B and η_T
Output: all maximal peculiar compositions in T against B

```
construct the generalized suffix tree ST for all strings in T, B
for v in postorder traversal of ST do
    store (f(BS(v)|T), f(BS(v)|B)) to v
end for
for v in postorder traversal of ST do
    if BS(v) is contrastive in target and f(BS(v)|T) ≥ η_T then
        for an ancestor u of v do
            if BS(u) is contrastive in background then
                isContrast(ST, u, v, θ_B) {find appropriate divisions for xy}
            end if
        end for
    end if
end for
```

Next FPCS performs a postorder traversal again, and calls `isContrast()` (see Algorithm 2) for nodes whose branching strings are candidates for peculiar compositions. The branching string $BS(v)$ is a candidate for a peculiar composition $xy = BS(v)$. The subroutine `isContrast()` checks if y is contrastive in background or not for all divisions (x, y) of xy. If so, it outputs xy as a peculiar composition. To check these combinations, the suffix link plays an important role in `isContrast()`.

Theorem 1. *FPCS finds all maximal peculiar compositions in T against B.*

Proof. FPCS traverses all nodes of the suffix tree constructed from T and B in postorder (the outer `for` loop in Algorithm 1) and then checks if the branching string is peculiar or not. We do not need to check a substring w which is not a branching string of the suffix tree since there exists a branching string w' such that it is peculiar and its prefix is w and we have to find only maximal if the substring is peculiar. Let v be a node in the traversal. $BS(v)$ is a candidate for a peculiar composition if it is contrastive in target and $BS(v) \geq \eta_T$.

Next we check if there exists a division (x, y) of $BS(v)$ such that both x and y are contrastive in background (`isContrast()`). We do not have to check all $|BS(v)| - 1$ divisions. Let u be an ancestor of v such that $BS(u)$ is contrastive in background (see Fig. 3). In this case, we have to check for $(x, y) = (a, bcdefgh), (ab, cdefgh)$, because a longer prefix of $BS(v)$, such as abc, is not contrastive in background.[3] Now we know that $BS(u)$ is contrastive in background and so the next problem is $bcdefgh$ or $cdefgh$ is contrastive in background or not.

Consider that FPCS visits a node w in Fig. 3 after some suffix link traversal. If $BS(w)$ is contrastive in background then FPCS outputs $BS(u)BS(w)$ as a peculiar composition. If not, then FPCS visits nodes above w because branching strings of these nodes might be contrastive in background, and if FPCS finds such a node w' then outputs $BS(u)BS(w')$ as a peculiar composition.

[3] However, we have to check for $(x, y) = (abcde, fgh), (abcdef, gh)$ when $u = \text{parent}(v)$.

Algorithm 2. The subroutine `isContrast()`

Input: ST: generalized suffix tree, u, v: node, θ_B: ratio
Output: **true** or **false**

```
if v = root(T) then return end if {root(T) returns the root of ST}
x := BS(u) {we know that x (resp. xy) is contrastive in background (resp. target)}
y := edge(u, v) {edge(u, v) returns the string label between u and v}
w := v {w is the start node of suffix link traversal}
for l := 1 to |BS(u)| do
  w := node pointed by the suffix link of w;
  if BS(w) is contrastive in background then
    print BS(u)BS(w) {Found!}
  else
    w' := w {w' is the start node of upword traversal}
    for k := 1 to |edge(v, u)| do
      k := k + |edge(parent(w'), w')| {parent(u) returns the parent node of u}
      w' := parent(w')
      if k > |edge(v, u)| then break end if
      if w' is contrastive in background then
        print BS(u)BS(w') {Found!}
        break {since upword nodes are not maximal}
      end if
    end for
  end if
end for
```

This procedure corresponds to the check for $y = cdefgh$, and then FPCS visits the next node followed by the suffix link and check for $y = bcdefgh$. Thus all possible divisions are checked. □

The time complexity of the proposed algorithm is given in the following theorem.

Theorem 2. *The time complexity of FPCS is* $O(N^2)$.

Proof. Construction and traversal of the suffix tree are done in linear time with respect to the total length of input strings [21,22].

To find appropriate divisions, we need to check u which is an ancestor of v (see Fig. 3). u corresponds to x of a peculiar composition. In addition to u, FPCS visits other nodes w and w'. These three nodes are different. Therefore, for each node v, FPCS visits at most $O(N)$ nodes.

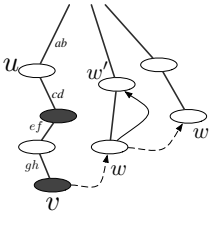

Fig. 3. Dotted arrows are suffix links and black nodes contain branching strings which are contrastive in target

At each node, FPCS tests if the corresponding branching string is contrastive or not if it is η_T-frequent. This tests can be done in constant time, using $(\mathrm{f}(v|B), \mathrm{f}(v|T))$ stored at all nodes. Thus the time complexity of the algorithm is $O(N^2)$. □

5 Experiments

We have implemented FPCS in C and conducted experiments on DNA sequences. All experiments were conducted on PowerMacG5 (OS: Mac OS X 10.5, CPU: 4×2.5GHz PowerPC G5, Memory: 8GB, and Compiler: gcc 4.0.1 with -O3, -arch ppc64 and -fast flags).

As input strings, we use DNA sequences. In particular, we often use the entire DNA nucleotide sequence of *Escherichia coli K-12* (RefSeq NC_00096, GI:50812173) and *Bacillus subtilis* (RefSeq NC_00096, GI:50812173). In addition to a sequence of GenBank, we add the complementary strand of the sequence into an input set for FPCS. For example, the input set of the entire DNA sequence of *E. coli K-12* consists of two strings, one is the original sequence and the other is its complementary strand. The sizes of these sets are 9279350bp (*E. coli K-12*) and 8429260bp (*B. subtilis*). Other sequences we used will be described at the corresponding experiments.

We first study the performance of FPCS with different data sizes or parameters in section 5.1 and then examine peculiar compositions output by FPCS from DNA sequences in section 5.2.

5.1 Performance Study

Firstly, we conduct two types of perfomance studies: one examines the relationship between the execution times and the number of peculiar compositions output by FPCS with different parameters, such as η_T or θ_B, and the other the execution times with different data sizes. For all experiments in this subsection, the background and target sets are fixed to the entire DNA sequence of *E. coli K-12* and one of *B. subtilis*, respectively.

Performance on Different Parameters. We examine the execution times and the number of peculiar compositions with different parameters, such as η_T or θ_B, for FPCS. Fig. 4 shows execution times of FPCS in second as η_T increase from 2 to 15, where $\theta_T = 5$ or 10 and $\theta_B = 1.5$, 2.0, 2.5 or 3.0. For any pair of parameters in Fig. 4, an execution time decreases as η_T increases. When θ_T or θ_B is large, the execution time seems not to decrease even if η_T increases. In such a case, however, the execution time does decrease drastically (see Fig. 5). Thus, pruning by η_T works effectively.

A decreasing rate becomes bigger as θ_T and θ_B decrease. These decreasing rates basically depend on θ_B but not on θ_T. In fact, we see every two lines for two values of θ_T with the same value of θ_B are close to each other.[4] Although θ_T and θ_B are not directly used to stop a traversal of FPCS, they can reduce the number of the candidates of xy, x or y. However, since there does not exist so many candidates of xy, θ_T does not decrease execution times effectively. On the other hand, we have many candidates for x and y, and thus θ_B can reduce the number of these candidates and execution times.

From these observations, we have the assumption that the execution time is closely related with the number of outputs. Two graphs in Fig. 5 show both the execution time (y-axis at the left-hand side) and the number of peculiar compositions output by FPCS (y-axis at the right-hand side), where $\theta_B = 1.5$ or 2.5.

[4] Exceptionally execution times are quite different when $\theta_B = 2.0$ and $1 \leq \eta_T \leq 5$.

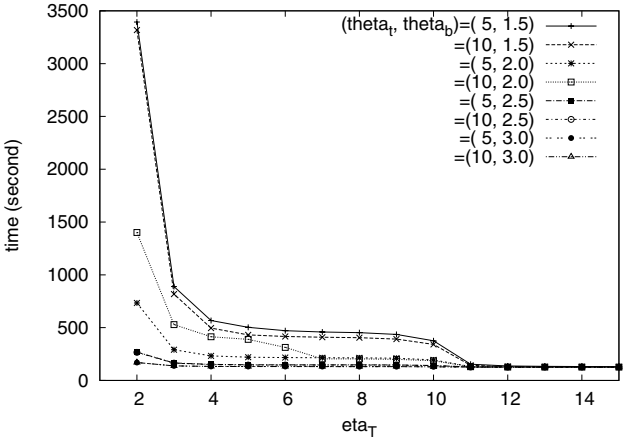

Fig. 4. Execution times of FPCS in second as η_T increases for various values of (θ_T, θ_B)

From these two graphs, we find that execution times are proportional to the number of peculiar compositions and the differences between $\theta_T = 5$ and 10 for a fixed η_T are negligible.

Scalability on Input Size. We examine the execution times with different data sizes. All execution times we show in this section are the average execution times of 5 trials on the same input.

As we increase the data size, we measure execution times of FPCS. To change the data size, we automatically generate m sequences with the same length n from sequences of *E. coli K-12* and *B. subtilis*, given m and n.

The graph on the left-hand side in Fig. 6 shows execution times of FPCS as n increases from 100 to 5000, where m is fixed to 1000, while execution times when m increases from 100 to 10000 in case $n = 500$ are given on the right-hand side.

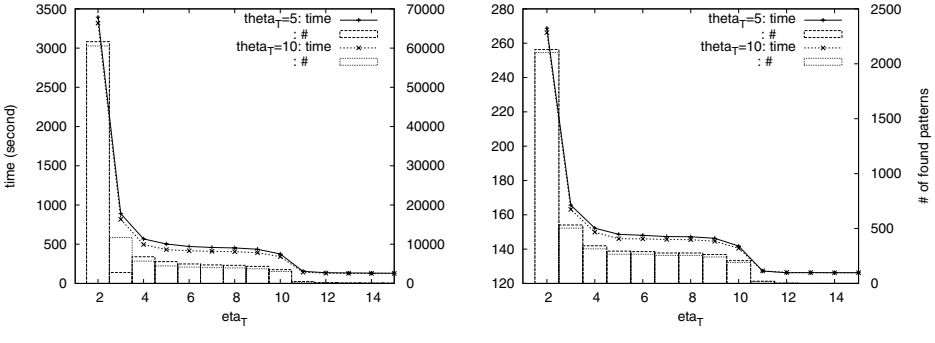

Fig. 5. Execution times of FPCS in second and the number of peculiar compositions output by FPCS as η_T increase in case of $\theta_B = 1.5$ (left-hand side) and $\theta_B = 2.5$ (right-hand side)

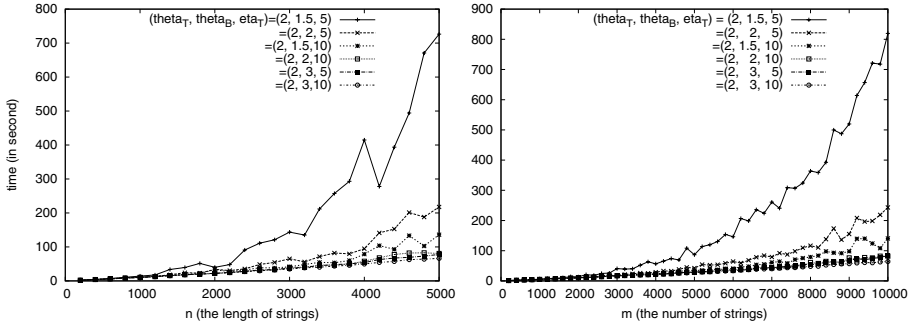

Fig. 6. Execution times for FPCS versus n (the length of strings) and m (the number of strings)

The execution times with 6 different sets of parameters, $(\theta_T, \theta_B, \eta_T) = (2, 1.5, 5)$, $(2, 1.5, 10)$, $(2, 2, 5)$, $(5, 2, 10)$, $(2, 3, 5)$, and $(2, 3, 10)$ are given. For any set of parameters, execution times in both graphs typically scale linearly with respect to the input size, except for $(\theta_T, \theta_B, \eta_T) = (2, 1.5, 5)$, although the time complexity of FPCS is $O(N^2)$ (see Section 4).

The increasing rate of the execution times in the right-hand side graph is larger than those in the left-hand side. This fact indicates that the execution time depends heavily on the depth of the suffix tree because the depth increases as the length n of the strings increases.

5.2 Properties of Peculiar Compositions

Secondly, we examine peculiar compositions from the entire DNA sequence of *E. coli K-12* or *B. subtilis*, as a target set using different background sets.

Peculiar Compositions in *B. Subtilis* against *E. Coli K-12*. We examine peculiar compositions in *B. subtilis* against *E. coli K-12* with different parameters. First, we set $(\theta_T, \theta_B, \eta_T) = (1.1, 5, 10)$ (see Table 1) and then $(10, 2, 15)$ (see Table 2), where one of θ_T or θ_B is small and the other is large.

Table 1 shows found peculiar compositions and related values, such as their frequencies, in case $(\theta_T, \theta_B, \eta_T) = (1.1, 5, 10)$. The z-score for each composition is also given.

Table 1. Peculiar compositions in *B. subtilis* against *E. coli K-12* with $(\theta_T, \theta_B, \eta_T) = (1.1, 5, 10)$. From left to right, prefix x and suffix y of compositions of the form xy, their lengths, the frequencies of xy, x, y in T and B, and z-scores are given. Probabilities of letters are computed over the whole sequences of the target set to compute the z-score.

(x, y)	length	(f(xy|T), f(xy|B))	(f(x|T), f(y|B))	(f(y|T), f(y|B))	z-score
$(CGGCGTGG, ACTACCAG)$	(8, 8)	(10, 7)	(66, 450)	(19, 154))	3.572e+02
$(CTGGTAGT, CCACGCCG)$	(8, 8)	(10, 7)	(19, 154)	(66, 450)	3.572e+02
$(GCGTGG, ACTACCAG)$	(6, 8)	(10, 7)	(529, 3845)	(19, 154)	7.759e+01
$(GGCGTGG, ACTACCAG)$	(7, 8)	(10, 7)	(161, 1407)	(19, 154)	1.666e+02

Table 2. Peculiar compositions in *B. subtilis* against *E. coli K-12* with $(\theta_T, \theta_B, \eta_T) = (10, 2, 15)$

(x, y)	length	$(f(xy\|T), f(xy\|B))$	$(f(x\|T), f(y\|B))$	$(f(y\|T), f(y\|B))$	z-score
$(CAGCG, GCGCC)$	(5, 5)	(17, 0))	(9816, 24161)	(6759, 17014)	8.924
$(GGCGC, CGCTG)$	(5, 5)	(17, 0)	(6759, 17014)	(9816, 24161)	8.924
$(CGCG, GCGCC)$	(4, 5)	(16, 1)	(16950, 56436)	(6759, 17014)	2.235
$(GGCGC, CGCG)$	(5, 4)	(16, 1)	(6759, 17014)	(16950, 56436)	2.235

To compute the score, we use the Bernoulli model where probabilities of individual letters, A, C, G, and T, are computed from the whole sequences of the target set and we have $p(A) = p(T) = 0.282$ and $p(C) = p(G) = 0.218$.

Four compositions are found as peculiar compositions and first two compositions are complementary each other. The lengths of the compositions are about 15. All of them appear 10 times in the target set while 7 times in the background set. These frequencies in the two sets are close since the given value for θ_T is nearly 1. However the peculiar compositions are unusual since frequencies of x and y in the background set are much larger than those in the target set.

The z-score is a normalized score whose average value is zero and variance is one [9]. Therefore, z-scores in Table 1 seem to be quite large and these composition are also unusual from the viewpoint of the z-score.

Next, we set $(\theta_T, \theta_B, \eta_T) = (10, 2, 15)$ for the same data sets. We have two pairs of two peculiar compositions which are complementary each other (see Table 2). All of found compositions appear 16 or 17 times in the target set while at most once in the background set.

Compared to those in Table 1, frequencies of x or y are quite large in Table 2 since we set a large value for θ_T and a small one for θ_B in Table 2 conversely. Compared to those in Table 1, the absolute values of the z-scores of found peculiar compositions are small. This means that, when we use the z-score to measure unusualness, it is difficult to find these compositions as unusual patterns because of the following reason. Since the z-score is a normalized score, we can calculate, for a positive number a, the number of substrings such that the absolute values of their z-scores are greater than $a = a\sigma$, where $\sigma = 1$. For example, in case $a = 3$, that number is 0.3% of the all substrings in given input data. Lengths of sequences used in the previous experiments, except for the performance study, are at least $N = 10^6$ and then we can estimate the number of all substrings by $O(N^2) = 10^{12}$. Hence, there exist a huge number of substrings whose z-scores are greater than $a = 2.235$.

Peculiar Compositions against *Saccharomyces Cerevisiae*. Now we examine compositions from the same data, *B. subtilis*, as a target set but against a set of sequences of *Saccharomyces Cerevisiae*, where we use sequences of Saccharomyces cerevisiae chromosome I–XVI (complete sequences) and Saccharomyces cerevisiae mitochondrion (complete genome) with their complementary strands. The total size of these sequences is about 24Mbp. The values of the parameters are set as $(\theta_T, \theta_B, \eta_T) = (20, 5, 20)$. 34 peculiar compositions are found from the target set and their lengths are about 14,

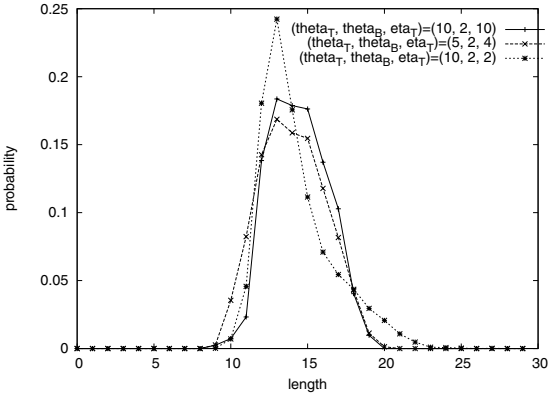

Fig. 7. The probability distributions for lengths of peculiar compositions for three parameter sets

ranging from 12 to 16. The z-scores for them are in the order of 10^1 or 10^2. We have many compositions with different divisions among 34 compositions. For example, we have three divisions, (x, y)=$(AAAACTAA, ACAAGACA)$, $(AAAACTAAA, CAAGACA)$ and $(AAAACTAAAC, AAGACA)$ for $xy = AAAACTAAACAAGACA$.

Next we try to find peculiar compositions from *E. coli K-12* against the same background set. 26 peculiar compositions are found, their lengths are about 13, the shortest one is 12 and the longest one is 16, and the z-scores are also in the order of 10^1 or 10^2. No composition of them is included in above 36 compositions.

Distribution of Lengths of Peculiar Compositions. In the previous experiments under many parameter settings, found peculiar compositions have similar lengths for some fixed parameter setting. Now we examine a distribution of their lengths.

Fig. 7 shows three probability distributions for lengths of found peculiar compositions under $(\theta_T, \theta_B, \eta_T)$=$(10, 2, 10)$, $(10, 2, 2)$ and $(5, 2, 4)$. Here, the probability means the relative frequency. We see that these distributions form bell shaped curves.

When we consider the z-score to measure unusualness, there exist many substrings whose lengths are quite small, such as 4 or 5. On the other hand, there are no short peculiar compositions of them. In fact, 9 is the shortest length of the peculiar compositions in Fig. 7. Moreover, this fact also holds when we found the small number of peculiar compositions while we have many short substrings according to the z-score even if we found the small number of substrings given a large threshold for the score. In fact, the number of found peculiar compositions is small and their lengths are not short in the previous experiments.

We think that FPCS collaterally discovers an appropriate model in which frequent substrings have similar lengths for a set of fixed parameters. This model is similar to the Markov model in which the estimation is calculated by substrings with length k. However, it is required to decide k in advance in the Markov model. On the other hand, FPCS simultaneously decides appropriate lengths in addition to finding peculiar compositions and their frequent components. Moreover, in our setting, substrings of peculiar compositions are not strictly restricted to have the same length k.

6 Conclusion

We have defined the peculiar composition discovery problem and then developed our algorithm, called FPCS, which runs in $O(N^2)$ time. Despite of its quadratic complexity, we have shown that FPCS typically runs linearly for many sets of parameters. We have also conducted experiments using DNA sequences and found, without assuming a probabilistic model, peculiar compositions which seem to be difficult to be found according to the z-score.

It is an important future work to apply our algorithm to other real data and to evaluate found peculiar compositions by domain experts. It is a challenging future work to consider more sophisticated patterns instead of compositions since a composition is defined by concatenation of only two strings without overlaps and hence we assume these strings occur independently. We think the MO method, introduced in [24], is promising for such a pattern. Using this method, we can estimate a probability of a given word w as follows: $p(w) = p(w_1)p(w_2|w'_1)\cdots p(w_k|w'_{k-1})$, where w'_i is a suffix of w_i. Note that the length of w'_i is variable. In [25], a linear time algorithm is presented to construct the model and to estimate probabilities for all given documents.

Acknowledgments

This research was partially supported by the Ministry of Education, Science, Sports and Culture, Grant-in-Aid for Scientific Research (B) 21300053 and Grant-in-Aid for Young Scientists (B) 19700150, and the Strategic International Cooperative Program funded by Japan Science and Technology Agency (JST).

References

1. Fayyad, U.M., Piatetsky-Shapiro, G., Smyth, P.: From Data Mining to Knowledge Discovery: An Overview. In: Advances in Knowledge Discovery and Data Mining, pp. 1–34. AAAI/MIT Press, Menlo Park, Calif (1996)
2. Knorr, E.M., Ng, R.T.: Finding Intensional Knowledge of Distance-Based Outliers. In: 25th International Conference on Very Large Data Bases, pp. 211–222 (1999)
3. Padmanabhan, B., Tuzhilin, A.: Small is Beautiful: Discovering the Minimal Set of Unexpected Patterns. In: Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 54–63 (2000)
4. Sarawagi, S., Agrawal, R., Megiddo, N.: Discovery-Driven Exploration of OLAP Data Cubes. In: Schek, H.-J., Saltor, F., Ramos, I., Alonso, G. (eds.) EDBT 1998. LNCS, vol. 1377, pp. 168–182. Springer, Heidelberg (1998)
5. Ide, T., Inoue, K.: Knowledge Discovery from Heterogeneous Dynamic Systems using Change-Point Correlations. In: 2005 SIAM International Conference on Data Mining, pp. 571–576 (April 2005)
6. Keogh, E., Lonardi, S., Chiu, B.Y.: Finding Surprising Patterns in a Time Series Database in Linear Time and Space. In: Eighth International Conference on Knowledge Discovery and Data Mining, pp. 550–556 (2002)
7. Apostolico, A., Bock, M.E., Lonardi, S., Xu, X.: Efficient Detection of Unusual Words. Journal of Computational Biology 7(1/2), 71–94 (2000)

8. Leung, M.Y., Marsh, G.M., Speed, T.P.: Over- and Underrepresentation of Short DNA Words in Herpesvirus Genomes. Journal of Computational Biology 3(3), 345–360 (1996)
9. Parida, L.: Pattern Discovery in Bioinformatics: Theory & Algorithms. Chapman & Hall Crc Mathematical & Computational Biology Series. Chapman & Hall/CRC, Boca Raton (2007)
10. Schbath, S.: An Efficient Statistic to Detect Over- and Under-represented Words in DNA Sequences. Journal of Computational Biology 4(2), 189–192 (1997)
11. Arimura, H., Shimozono, S.: Maximizing agreement with a classification by bounded or unbounded number of associated words. In: Chwa, K.-Y., Ibarra, O.H. (eds.) ISAAC 1998. LNCS (LNAI), vol. 1533, pp. 39–48. Springer, Heidelberg (1998)
12. Ikeda, D.: Characteristic Sets of Strings Common to Semi-structured Documents. In: Arikawa, S., Furukawa, K. (eds.) DS 1999. LNCS (LNAI), vol. 1721, pp. 139–147. Springer, Heidelberg (1999)
13. Ji, X., Bailey, J., Dong, G.: Mining Minimal Distinguishing Subsequence Patterns with Gap Constraints. In: Fifth IEEE International Conference on Data Mining, pp. 194–201 (2005)
14. Nakanishi, M., Hashidume, M., Ito, M., Hashimoto, A.: A Linear-Time Algorithm for Computing Characteristic Strings. In: Du, D.-Z., Zhang, X.-S. (eds.) ISAAC 1994. LNCS, vol. 834, pp. 315–323. Springer, Heidelberg (1994)
15. Suzuki, E.: Autonomous Discovery of Reliable Exception Rules. In: Third International Conference on Knowledge Discovery and Data Mining, pp. 259–262 (1997)
16. Suzuki, E.: Undirected Discovery of Interesting Exception Rules. International Journal of Pattern Recognition and Artificial Intelligence 16(8), 1065–1086 (2002)
17. Suzuki, E., Shimura, M.: Exceptional Knowledge Discovery in Databases Based on Information Theory. In: Second International Conference Knowledge Discovery and Data Mining, pp. 275–278 (1996)
18. Suzuki, E., Tsumoto, S.: Evaluating Hypothesis-Driven Exception-Rule Discovery with Medical Data Sets. In: Terano, T., Chen, A.L.P. (eds.) PAKDD 2000. LNCS, vol. 1805, pp. 208–211. Springer, Heidelberg (2000)
19. Wang, J., Han, J., Pei, J.: CLOSET+: Searching for the Best Strategies for Mining Frequent Closed Itemsets. In: 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 236–245 (2003)
20. Yan, X., Han, J., Afshar, R.: CloSpan: Mining Closed Sequential Patterns in Large Databases. In: The 4th SIAM International Conference on Data Mining (May 2003)
21. McCreight, E.M.: A Space-Economical Suffix Tree Construction Algorithm. Journal of the ACM 23(2), 262–272 (1976)
22. Ukkonen, E.: On-line Construction of Suffix Trees. Algorithmica 14(3), 249–260 (1995)
23. Gusfield, D.: Algorithms on Strings, Trees and Sequence. Cambridge University Press, New York (1997)
24. Jagadish, H.V., Ng, R.T., Srivastava, D.: Substring Selectivity Estimation. In: Eighteenth Symposium on Principles of Database Systems, pp. 249–260 (1999)
25. Uemura, T., Ikeda, D., Arimura, H.: Unsupervised Spam Detection by Document Complexity Estimation. In: Boulicaut, J.-F., Berthold, M.R., Horváth, T. (eds.) DS 2008. LNCS (LNAI), vol. 5255, pp. 319–331. Springer, Heidelberg (2008)