Detecting Blog Spams using the Vocabulary Size of All Substrings in Their Copies

Kazuyuki Narisawa
Department of Informatics
Kyushu University
Hakozaki 6-10-1
Fukuoka 812-8581, Japan
k-nari@i.kyushu-u.ac.jp

Yasuhiro Yamada*
Department of Informatics
Kyushu University
Hakozaki 6-10-1
Fukuoka 812-8581, Japan
yamada@usi.kyushu-u.ac.jp

Daisuke Ikeda
Kyushu University Library
Hakozaki 6-10-1
Fukuoka 812-8581, Japan
daisuke@lib.kyushu-u.ac.jp

Masayuki Takeda
Department of Informatics
Kyushu University
Hakozaki 6-10-1
Fukuoka 812-8581, Japan
SORST, Japan Science and Technology Agency
takeda@i.kyushu-u.ac.jp

ABSTRACT

This paper addresses the problem of detecting *blog spams*, which are unsolicited messages on blog sites, among blog entries. Unlike a spam mail, a typical blog spam is produced to increase the PageRank for the spammer's Web sites, and so many copies of the blog spam are necessary and all of them contain URLs of the sites. Therefore the number of the copies, we call it the *frequency*, seems to be a good key to find this type of blog spams. The frequency is not, however, sufficient for detection algorithms which detect an entry as a blog spam if the frequency is greater than some threshold value, because of the following reasons: it is very difficult to collect Web pages including all copies of a blog entry; therefore an input data contains only a few copies of the entry whose number may be smaller than the predefined threshold; and thus a frequency based spam detection algorithm fails to detect. Instead of frequency based approaches, we propose a spam detection method based on the *vocabulary size*, which is the number of substrings whose frequencies are the same. The proposed method utilizes the fact that the vocabulary size of substrings in normal blog entries follows the Zipf's distribution but the vocabulary size in blog spams does not. We show its effectiveness by experiments, using both artificial data and Web data collected from actual blog entries. Experiments using Web data show that the proposed method can detect a blog spam even if the frequency of it is not so large, and that the method finds all blog spams with some copies simultaneously in given blog entries. A blog spam written in Chinese, which seems to be advertisements for Chinese movies, is found from an English blog site. This result shows that the proposed method is independent from the language. We also show the scalability of the proposed method with respect to input size using a huge size of text data.

*The current affiliation is the User Science Institute, Kyushu University, Hakozaki 6-10-1

.

Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: Retrieval models; I.5.4 [**Applications**]: Text processing

General Terms

Algorithm, Experimentation, Performance

Keywords

Spam Detection, Outlier Detection, Zipf's Law, Power Law

1. INTRODUCTION

A blog is a frequently updated journal on a Web site. A blog software, such as Movable Type, provides the basic function for blog users to write their journals easily: they only write their journals, and do not need to know any HTML tags to format and sort their journals. And a blog software also provides communication functions among blog users and readers, such as comments and track backs. By the simplicity and the communication functions of the blog, the number of blog sites are increasing rapidly. However, these functions cause the problem of *blog spams*.

Blog spams are unsolicited bulk messages, also known as comment spams, link spams, splogs, etc[6, 9]. Blog spams usually contain URLs of spammers' Web sites or blog sites, and they intend to manipulate PageRanks and other similar metrics of these sites for search engine optimization (SEO).

The existing methods to decrease effects of spams are categorized into three types:the regulation, the link structure analysis, and the contents analysis.

Regulation: The *no follow tag*, which is a kind of regulation, is introduced by search engine companies [5]. This tag is used as follows: only the administrator or the owner of a blog site can use the tag; links only above this tag are used to calculate PageRanks or other similar metrics, while the links below the tag are ignored; and therefore the effect of blog spams is reduced. The no follow tag, however, is not effective if it is not used widely. Even if it is used widely, any links even in normal comments or track backs below this tag are also ignored.

Link structure analysis: Link farms are dense clusters of Web pages with mutual links to raise PageRanks each other. Some methods to find them using the link structure analysis are proposed [2, 3, 18]. It is, however, difficult to judge automatically whether the mutual links are malicious links or not.

Contents analysis: This type method, which is used for spam mail detection, discovers differences in appearances of letters or words between spams and non-spams [4, 7, 15, 17]. A mail filtering software learns classification rules using training data [4, 7, 17]. It is costly to make training data. A similar method is also developed for blog spams [15]. However, the main purpose of spams is to give many links to spammers' Web sites, and so contents of them are simple, natural sentences, unlike spam mails. Therefore, it seems to be difficult to find significant differences among blog spams and others.

The goal of this paper is to develop a method detecting blog spams. To achieve this goal, first consider what blog spams are minutely. A blog entry is generally said to be a spam if its purpose is the advertisement or to manipulate the PageRank, or if its content is wicked. This definition is vague and it is very difficult to detect such a vague target automatically. Instead of purposes of blog entries or spammers' intentions, we pay attention to the cost of spammers.

Spammers must create many copies of an entry at low cost to achieve their purposes. So when we want to detect blog spams, the number of copies of a blog entry, that is, the frequency of the entry, seems to be an important point. The simple frequency of the entry, however, is not effective for spam detection, because we cannot collect all blog entries on the Web and so collected data may not have enough copies of a blog spam to detect it as a spam. Thus, the frequency based method may not work.

Therefore, threshold for frequency based approaches, such as a spam mail detection algorithm in [19], is not applicable perfectly to the spam detection. We need another measure instead of the simple frequency.

In addition to the frequency of spams, we consider the length of blog entries as an important key to detect blog spams. Contents of blog spams must be long enough, because the purpose of the spams are an advertisement or SEO, and they contain URLs of spammers' sites. In other words, only few words or phrases cannot deliver spammers' messages.

In this paper, we propose a unique spam detection method for spam detection based on the *vocabulary size* of all substrings in blog entries. The vocabulary size is the number of strings whose frequencies are the same[1] The basic idea of the proposed method is the Zipf's law, which states that the vocabulary size is inversely proportional to the frequency of strings in natural language sentences [20, 21]. In other words, the Zipf's law says that a *size-frequency plot*, which is a plot of vocabulary sizes for each frequency in double-logarithmic scale, is a straight line with negative slope. So, the frequency of substrings in non-spams follows a Zipf's distribution, while that in blog spams does not, because the existence of many copies of a spam increases the vocabulary size abnormally.

[1]Generally the vocabulary size is defined by the words, but note that our definition of the vocabulary size is not based on words but substrings of blog entries.

Some of the authors developed similar method, called the substring amplification, using the *total occurrence* instead of the vocabulary size [10, 11]. It was developed to discover common strings and templates in Web pages. The substring amplification, however, has serious weak points when we use this as a spam detection algorithm. First, we must see the total occurrence-frequency plot and find the remarkable spikes, which is constituted by common strings. The linear time complexity of the substring amplification goes away because of this operation. Second, a common string, such as a template of Web pages, appears most of input strings, but a spam may appear a few times in the input blog entries even if many copies of the spam may exist on the Web. We do not know that the substring amplification can detect such spams with a few copies. Third, the substring amplification can not distinguish spams with other high frequent words, which includes non-templates or non-spams, though the spams are also included in frequent words.

We suggest a conditional expression for the vocabulary size, instead of the total occurrence, to develop an automatic algorithm for spam detection. The vocabulary size decreases monotonically as the frequency increases, although the total occurrence does not. Thus, it is natural to make the conditional expression based on the vocabulary size. This conditional expression is a key point for an automatic spam detection algorithm.

In the spam detection problem, both the false-negative and false-positive problems are so important. The proposed method resolves the false-negative problem, while it is difficult to solve the false-positive problem by our method, because our method does not refer contents of detected strings. The false-positive problem is also difficult for other spam detection algorithms if these algorithms work automatically, since (blog) spams are not defined clearly.

We conduct four experiments. First, we show the scalability of the proposed method. Our implementation of it is scale linearly, and so it can process about 1.5G bytes text data in about 30 minutes. Next, we estimate the lower bound of the length and the frequency of the detectable spams experimentally using artificial data. As mentioned above, the longer or more frequent spams are, the easier their detection is. Next, we estimate the lower bound of the number of copies of one message empirically. Obviously, it is easy to detect many copies, but we do not know the lower bound. In other words, these two experiments show that the proposed method does not have the false-negative problem if blog spams have some adequate length. Finally, we show the applicability of our algorithm to real data on the Web. The data, where we do not know that include spams or not, is from the Internet blog site "Arianna's Blog". We found several spams written in some languages.

2. SPAM DETECTION USING SUBSTRING AMPLIFICATION

In this section, we first explain the substring amplification, which finds the common string, according to [10, 11], and then point out some serious problems we are faced with when

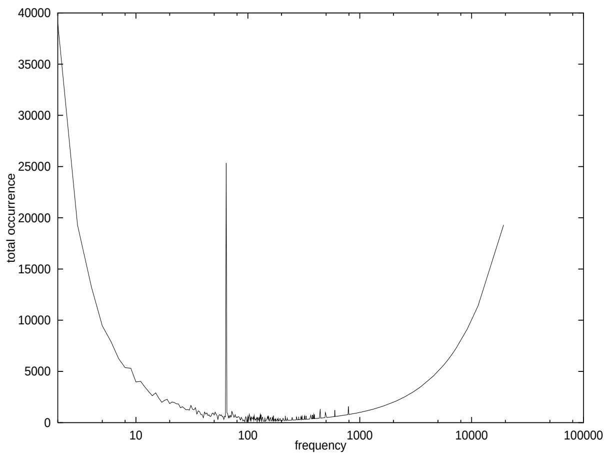


Figure 1: Relation between the frequency f and the total occurrence $T(f)$ for artificial data with 64 spams. The x-axis is the frequency in log scale, and the y-axis is total occurrence. The remarkable spike at $f = 64$ is constituted by long substrings of the common string

applying this algorithm to the spam detection problem.

Let Σ be a finite alphabet, and let Σ^* denote the free monoid over Σ. An element of Σ^* is called a *string*. A *sample* is a finite subset S of Σ^*.

Suppose that all the strings in a sample S contain a common string w of length n, and other parts of strings are natural sentences. For simplicity, we now assume that w appears only once in each string in S, and so the frequency of w is $|S| = f$. In the spam detection problem, S is a set of HTML pages, each string in S corresponds to a blog entry with comments and track backs, and w is a blog spam included in all entries. We can assume that w is not so short, because it contain several sentences to lead readers to URLs of spammer's Web/blog sites.

We count frequencies of the substrings of w in S. First, the frequency of length n, which equals to the frequency of w itself, is obviously f. Conversely, we cannot find other string with frequency f, because $f = |S|$ is irregularly high frequency and other parts of strings in S are natural sentences. Next, we count shorter substrings. The frequencies of the substrings of length $n - 1$ are also f with high probability, because we can hardly find other strings with frequency f, which is irregularly large. The frequencies of the substrings of length $n-2$ are also f with high probability because of the same reason. Similarly, while the length of substrings of w is long enough, their frequencies are also f with high probability. Hence, the number of substrings whose frequencies are f is approximately $O(n^2)$. Therefore, we can expect that the total occurrence, denoted by $T(f)$, of substrings with frequency f is quite large.

To find common strings, the substring amplification finds some fs providing irregularly large $T(f)$. Figure 1 shows a relation between the frequency f and the total occurrence $T(f)$. We call such a graph a *total occurrence-frequency plot*. We see the remarkable spike at $f = 64$ whose total occurrence is constituted by a common string and many of its substrings. So, we can find the common string using $T(f)$.

To get a total occurrence-frequency plot, we need to enu-

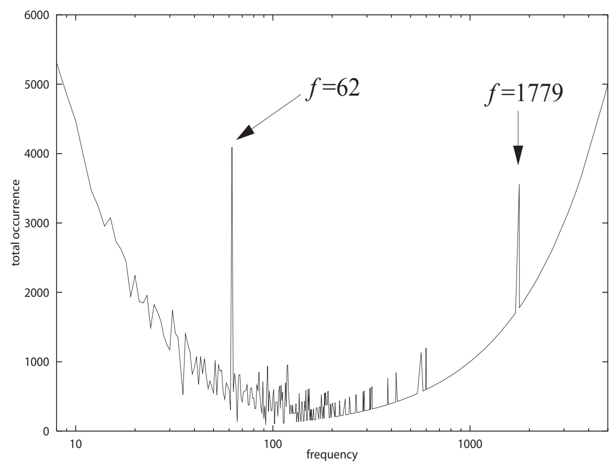


Figure 2: Relation between the frequency f and the total occurrence $T(f)$ for artificial data with 62 spams. The x-axis is the frequency in log scale($8 < f < 5000$), and the y-axis is total occurrence. The remarkable spike at $f = 62$ is constituted by long substrings of the common string, and the remarkable spike at $f = 1779$ is constituted by non-common strings

merate all substrings. Although there exist $O(n^2)$ substrings for a string of length n, an excellent data structure, such as the suffix tree [8], enables to count them in linear time.

Although the substring amplification is scalable and good to find common strings in general, it has three serious problems when applying to the spam detection problem.

First, we must see a total occurrence-frequency plot and find remarkable spikes manually. This problem is most serious, because it takes long time and judgments are very vague. In addition, manual judgments prone to misjudge such that we judge a remarkable spike with a non-common string as a common string. For example, in Figure 2 we can find two spikes, one is the spike constituted by a common string at $f = 62$, the other is the spike constituted by a non-common string at $f = 1779$, because the substring amplification uses the total occurrence $T(f)$. Total occurrence is large if strings are high frequency or a string is long. So total occurrence is large if strings are high frequency but a string is not long enough. Thus, we need an automatic algorithm to detect spams and not to misjudge.

Second, we do not know whether the substring amplification is applicable to the spam detection problem. The substring amplification was originally developed to detect common strings, such as templates in HTML text on the Web [11]. The number of templates in input data is generally similar to that of the file in input data. In addition, templates are long enough because they include long HTML tag sequences, URLs, and so on. So templates are high frequency and long enough. Spams, however, may not be long, compared to Web templates, and may not be frequent in input data. In other words, spams are hidden in huge amounts of non-spams, so that we need to find a small amount of spams from huge amounts of input data.

Third, in spam detection problem, there are the false-positive and false-negative problems. In general, spams are not defined exactly, so any spam detection methods needs

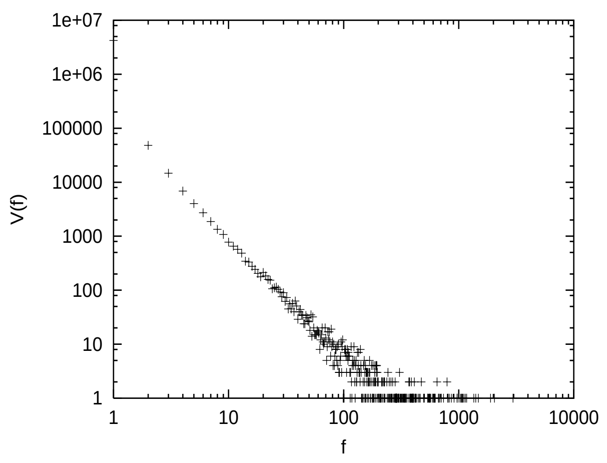


Figure 3: Relation between the frequencies f and the vocabulary sizes $V(f)$ for the novel "kokoro" in Japan. The x-axis is the frequency, and the y-axis is the vocabulary size in log scale. The vocabulary sizes are for the substrings frequencies

finally human judgment whether detected strings are spams or not. For a false-positive problem, the substring amplification may be effective, because it may be able to detect not only spams and templates but also non-spams and non-templates.

3. SPAM DETECTION BASED ON ZIPF'S LAW

In this section, we introduce a developed method which detects spams. The developed method is based on the Zipf's law.

3.1 Zipf's Law

Let $V(f)$ be the number of strings whose frequencies are f in a given sample S. We call $V(f)$ a *vocabulary size*. It is also known as the frequency of frequency in the natural language processing. Using the vocabulary size, the total occurrence is denoted by $T(f) = f \times V(f)$.

Between f and $V(f)$, the so-called Zipf's law holds [20, 21]: the vocabulary size is inversely proportional to the frequency of strings in natural language sentences. In other words,

$$\begin{aligned} V(f) &= bf^{-a}(a > 0), \\ \log V(f) &= \log b - a \log f. \end{aligned}$$

Therefore a *size-frequency plot*, which is a graph of the vocabulary size for each frequency, is a straight line with negative slope(show Figure 3).

Readers note that the above law is known as Zipf's second law, and that it is a statement about English words, not about (sub)strings. It is, however, shown empirically that the vocabulary size for substring frequencies also follows a Zipf distribution [10, 11].

Similar distributions are also found in words in other languages, links of WWW [1], product sales.

3.2 Conditional Expression

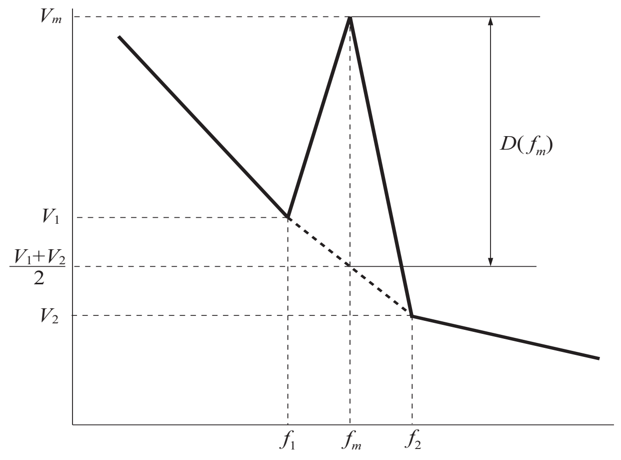


Figure 4: Conditional expression $D(f)$ is the difference between V_m on f_m and the median of V_1 and V_2 which are on f around f_m

The substring amplification is based on the total occurrence, which equals the product of the frequency and the number of the substrings. The total occurrence leads us to a misjudgment such that a remarkable spike is constituted by non-spams. A sample with spams has also the power-law distribution of the number of the substrings. Spams with enough length are more intended spams than spams with high frequency. Thus, we introduce a spam detection method with the vocabulary size for the substrings. In order to find remarkable spikes automatically, we must analyze the spikes numerically.

The remarkable spike is out of the natural distribution. In other words, if there is a remarkable spike at the frequency f, $V(f)$ is much lager than both $V(f-1)$ and $V(f+1)$. Hence, if there is a remarkable spike at the frequency f, $V(f)$ is under the following rule:

$$V(f-1) < V(f), V(f) > V(f+1).$$

We consider a remarkable spike itself. First, we estimate the height $V(f)$ of a remarkable spike from the number of the substrings at f. Because the number of the substrings of a string of length n is at most $n(n+1)/2$, we can estimate the height as $O(n^2)$.

Second, we estimate the number of the substrings of $V(f)$ when there is no remarkable spike at the frequency f. We consider that the distribution of the number of the substrings is monotonically decreasing. $V(f)$, the number of the substrings at the frequency f, exists between $V(f-1)$ and $V(f+1)$. Thus, we assume that the original number of the substrings $V(f)$ at the frequency f is $(V(f-1)+V(f+1))/2$.

Let f_m be a frequency f. Let f_1 be a frequency $f-1$ and f_2 be a frequency $f+1$. Then, V_1, V_m and V_2 is the value V on f_1, f_m and f_2. Suppose that V_m makes a remarkable spike, the difference on f_m is the following:

$$D(f) = V_m - \frac{V_1 + V_2}{2}.$$

Figure 4 is the visual graph of $D(f)$. We consider $D(f)$ as a conditional expression. We find spams by the conditional expression $D(f)$ for the frequencies, when $D(f)$ is large.

This conditional expression $D(f)$ judges the remarkable

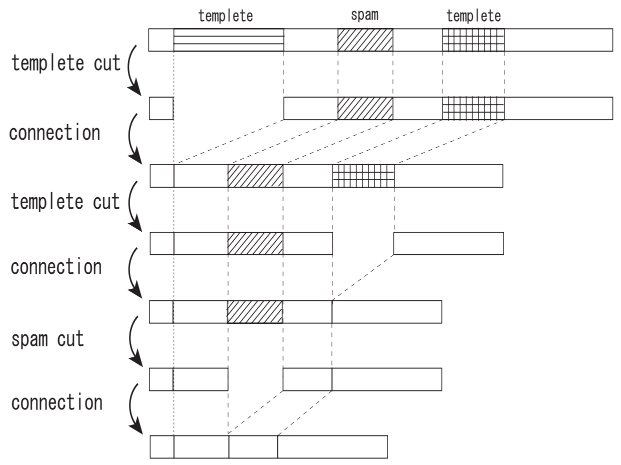


Figure 5: The spam detection algorithm can make non-spam data by the operation which the detected strings, spams or templates, are removed from dataset.

spike by the number of the substrings $V(f)$ instead of the total occurrence $T(f)$. The substring amplification uses the total occurrence, but in really this conditional expression $D(f)$ can find the remarkable spike using only the number of the substrings. In the substring amplification, the total occurrence is used because the spike is remarkable for the human seeing, thus we do not need the total occurrence essentially. This conditional expression can lead us to find the remarkable spike, which the substring amplification did not refer.

3.3 Algorithm

In preceding subsection, we showed conditional expression which can detect frequent strings automatically. However, we cannot detect several frequent strings, because the frequency f which $D(f)$ is the largest includes several kinds of strings and the maximum of $D(f)$ can detect only one spam. So, we operate remove the detected strings in addition to the conditional expression. This operation enable to detect several kinds of frequency strings, several kinds of strings on the same frequency and templates.

We can differentiate templates from spams easily, thus we can detect spam easily. Hence, this operation enable to make the data set which has no spams. Figure 5 shows the sensuous overview of the spam detection algorithm by this operation. Consequently, we have an algorithm for the spam detection problem (see Figure 6). Figure 5 is visual for this algorithm.

`Count(T)` is a function which enumerates the number of the substrings in sample `S`, where $T = w_1\$w_2\cdots w_n$, $S = \{w_1, w_2, \ldots, w_n\}$, and \$ is the special symbol not in Σ (see Figure 7). `Count(T)` uses a suffix tree for T.

Let v be a node in a suffix tree, u be its parent node, c be the number of leaves below v. The branching string for v is the string obtained by concatenating all strings labeled on edges of the path from the root to v. We denote by $|v|$ the length of the branching word v. With the suffix tree, all we have to do is to count only branching strings.

However, a suffix tree requires a much working space. Therefore `Count()` uses a suffix array instead and utilizes

```
function DiscoverString(var S: sample):sample
var
  OCC: hash(key:integer, value:integer);
  V: hash(key:integer,
          value:list of tuples of integers);
begin
  T:=w1$w2$...$w$; // S=w1, w2, ..., wn
  (OCC, V):=Count(T);
  for f in keys(V) do begin;
    for w in V{f} do begin;
    //w is denoted by (c, l, h) of integers
      if(|w-1|<|w| & |w|>|w+1|);
        D(f) = |w| - (|w-1|+|w+1|)/2;
      else;
        D(f) = 0;
    end;
  end;
  maxf = 0;
  maxD = 0;
  for f in (D(f)!=0) do begin;
    if(maxD < D(f))
      maxD = D(f);
      maxf = f;
  end;
  return(V(maxf));
end;

Algorithm SpamDetectionAlgorithm(Var S:sample)
begin
  while(spam ∈ S) do begin;
    string = DiscoverString(S);
    remove(string,S,newS);
    S = newS;
  end;
end;
```

Figure 6: Spam Detection Algorithm

the subroutine `TraverseWithArray()` [12] which computes the postorder traversal of a suffix tree with a suffix array.

To compute $V(f)$ for all frequencies, first `Count()` calculate total occurrences $T(f)$ using $(|v| - |u|) \times c$ for all nodes in a suffix tree. `Count()` outputs two hashes OCC and SEQ each of whose key is a frequency f. `H_f-H` equals to $T(f)$, so we have $V(f) =$ `OCC(f)`.

The value of `SEQ` is a list of tuples `(l1, h1, h2)`, which are integers, to express substrings with the same frequency in the given string. `Count()` uses two stacks `node` and `freq` which consist of tuples `(c, l, h)` of three integers. `node` is used for the postorder traversal, and `freq` is used for counting total occurrences $T(f)$. In the postorder traversal using a suffix array `SA`, each node of a suffix tree is expressed by a tuple `(c, l, h)`, where `c` means the number of leaves below the node, `l` is the position in a suffix array for the branching string of the node, and `h` is the length of the branching string. The substring of length `h` starting at `SA[l]` in a given string is the branching string of the node.

For each node `v` in the postorder traversal, `child` in Figure 7 represents the number of children nodes of `v` at the position `loop(1)`. The top `child` tuples in `freq` are the information about the all children nodes of `v` at `loop(1)`. `Count()` calculates $T(f)$ of the branching strings of the chil-

```
function Count(var T: string):
   hash(key:integer, value:integer),
   hash(key:integer,
        value:list of tuples of integers);
  var
    OCC: hash(key:integer, value:integer);
    SEQ: hash(key:integer,
     value:list of tuples (l1, h1, h2) of integers);
    SA: array[1...|T|] of integer; //suffix array
    Hgt: array[1...|T|+1] of integer; //height array
    node, freq: stack of tuples (c, l, h) of integers;
    L, H, L_i, H_i, H_f, child, f, i, j: integer;
  begin
    SA:=Suffix_Array(T); //Compute the suffix array
    Hgt:= Fast_Hgt(T, SA); //Compute the height array
    Push((0, -1, -1), node);
    for i:=1 to |T|+1 do begin
      L_i:=i-1; H_i:=Hgt[i];
      (child, L, H):=(c, l, h) of the top tuple in node;
      while (H > H_i) do begin
        Pop(node);
        for j := 1 to child do begin //loop(1)
          (f, H_f):= (c, h) of the top tuple in freq;
          OCC{f} += (H_f-H)/f;
          Add((SA[L], H_f, H), SEQ{f});
          Pop(freq);
        end; //for
        if (child = 0) then Push((i-L, L, H-1), freq);
        else then Push((i-L, L, H), freq);
        L_i:=L;
        (child, L, H):=(c, l, h) of the top tuple in node;
        child++;
       end; //while
       if (i = 1) then Push((0, 0, 0), node);
       else if (H < H_i) then Push((1, L_i, H_i), node);
       else then Increment c of the top tuple in node;
       Push((0, i, |T|-SA[i]), node);
     end; //for
     return (OCC, SEQ);
   end;
```

Figure 7: Count() enumerates all branching strings using the suffix array and calculates vocabulary sizes for all frequencies

dren nodes of v using their tuples `(c, l, h)`.

We estimate the time complexity of `Count()`. Let n be the total length of a given sample. `TraverseWithArray()` needs the suffix array of the string and its height array. A suffix array is computed in linear time The height array is also computed in $O(n)$ time, thus `TraverseWithArray` works in $O(n)$ time [12].

A tuple (c, l, h) for each node in a suffix tree is pushed on *freq* exactly once. The number of nodes in a suffix tree is at most $O(n)$, therefore, the $loop(1)$ in Figure 7 repeats exactly n times in total. Thus, `Count()` works in $O(n)$ time. Thus, it is trivially that this spam detection algorithm is in linear time. This algorithm returns `V(maxf)`, so if we want to detect some spams, we must repeat this algorithm.

The substring amplification has wrong spikes if the frequency is large. But this spam detection algorithm does not have such wrong spikes even if the frequency is large and the number of the substrings is tiny. In this algorithm, $f-1$ and $f+1$ may be not successive for f. If f is so large, the successive $f-1$ and $f+1$ does not exist in high probability. We need not consider the nonexistence because spams do not exist on large noncontiguous frequency.

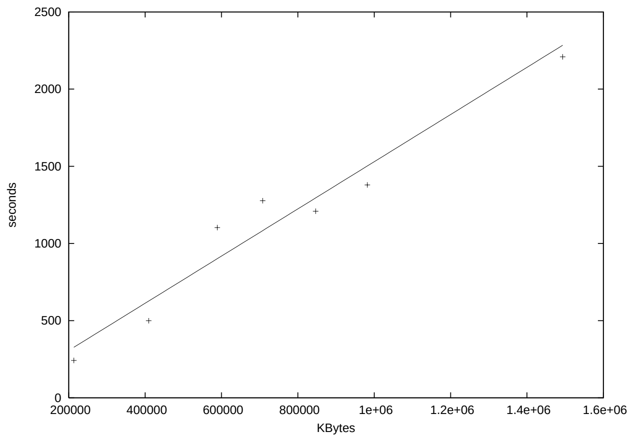


Figure 8: Consumption time by counting all substrings is showed in seconds. Data size is about from 200M to 1.5G bytes. This scalability is for one spam detection, not for SpamDetectionAlgorithm in Figure 6

This algorithm enables us to detect the spams automatically without human efforts. But this algorithm cannot know lower bounds of the length and the number of copies of the spams. Thus, in the next section, we examine the bounds and capability of a real spam detection on the Web.

4. EXPERIMENTS

We implemented the algorithm in Figure 6[2] on IBM eServer p5 model 595[3]. Using this implementation, we show the scalability and the applicability of the spam detection algorithm.

4.1 Scalability

Figure 8 shows that the time consumed by `Count()` routine is linearly proportional to the input size, where input data size is about from 200M to 1.5G bytes. This scalability is for one spam detection, not for all spams detection. The time required for about 1.5G bytes messages is about 30 minutes.

4.2 Lower Bounds of the Length and the Frequency of Spam Messages

It is easy to detect long and frequent spams, but we do not know how short spams or how few copies of a spam can be detected by the spam detection algorithm. In this section, we estimate the lower bounds of the length and frequency of detectable spams.

We prepare 2350 samples whose sizes are fixed to 100 messages. And the sizes of all messages are also fixed to 100 letters. In these fixed samples, we embed copies of one spam into some of 100 messages, where both the length of it and the number of its copies are varied among samples.

[2]Some linear time algorithms to construct a suffix array have been proposed, such as [13]. However, our implementation uses `deep-shallow()` [14] whose worst-case time complexity is $O(n^2 logn)$, because it is faster than other algorithms in practice.

[3]Although eServer p5 model 595 has 416 CPUs, we used only one CPU.

Table 1: The probability for appearances of alphabet letters in English sentences

letter	probability	letter	probability
a	0.0668	o	0.0654
b	0.0118	p	0.0162
c	0.0226	q	0.0010
d	0.0310	r	0.0559
e	0.1073	s	0.0499
f	0.0239	t	0.0856
g	0.0163	u	0.0201
h	0.0431	v	0.0075
i	0.0519	w	0.0126
j	0.0011	x	0.0014
k	0.0034	y	0.0162
l	0.0278	z	0.0006
m	0.0208	space	0.1817
n	0.0581		

The number of copies of the spam is 2, 4, ..., or 100, and the length of the spam is 4, 5, ..., or 50. Thus, we have $50 \times 47 = 2350$ samples.

All messages including the spam are pseudo English sentences which are generated randomly. We use Table 1 of [16] as the probabilities of the letters.

Figure 9 is the result with the spam detection algorithm. Each row corresponds to the length of the spam, and each column does to the number of messages containing copies of it. Black cells mean that the spam cannot be detected, while white cells mean the spam can be detected. We call a white cell a detectable one. We know the frequency f_c of the spam, so we can judge whether f providing the maximal $D(f)$ equals f_c or not.

From Figure 9, we know that the algorithm detects a spam if its length is larger than 10.

We also conducted the same experiment using the substring amplification. The number of detectable cells by the spam detection algorithm is 2054, while the number of detectable[4] cells the substring amplification is 2140. The spam detection algorithm detected less spams than the substring amplification, because we misjudged the frequency of the remarkable spike in the substring amplification. In other words, even if the remarkable spike is at $f-1$, we tend to consider that the spike is at f, because we know the frequency of the spam in advance. This is due to the imprecision of spike judgment by the human seeing.

Another significant difference among the substring amplification and the spam detection algorithm is time required by them. The substring amplification took about several hours because we have to see all graphs. On the other hand, the spam detection algorithm took only a few minutes.

4.3 Lower Bound of the Number of Copies of Spam Messages

On popular online blog and forum sites, there exist a huge number of messages. In such a situation, we must detect spams, though the number of copies of one spam might be relatively small, because we cannot collect all messages. In this section, we examine the lower bound of the number of copies of spams. In addition, we compare the spam detection algorithm with the substring amplification.

[4]In this case, the decision whether a cell is detectable or not is made by the authors subjects

Table 2: Spams in DataSet2

length	number
20	50
30	100
40	101
50	102
30	150

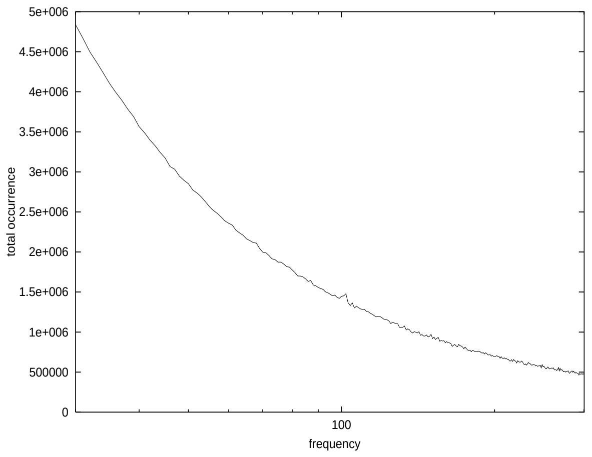


Figure 10: The total occurrence values $T(f)$ for the sample which have 200000 messages, where x-axis is frequency in log scale $(30 \leq f \leq 300)$

We fix the set of 5 spams whose length and the number of spams are defined in Table 2. In a sample, 50 (resp. 100, 101, 102, and 150) messages contain the same copies of the spam of length 20 (resp. 30, 40, 50, and 30).

There are 7 samples which contain 1000, 10000, 30000, 50000, 80000, 100000 and 200000 messages. All samples the same number of spams, and so the number of messages which do not contain the spams are different. A message has 1000 letters. We use the same probabilities in Table 1.

We examine whether a spam can be detected in the case that the number of the total messages is much larger than that of spams. Figure 10 is a result graph by 200000 messages. This sample has 102 spam whose length is 50, thus the spike should appear at $f = 102$. We cannot find any spike in the total occurrence-frequency plot, where $0 \leq f$. Even if we zoom in around $30 \leq f \leq 300$ (Figure 10), the spike is not so remarkable.

In contrast, the spam detection algorithm can detect the spam. A result of the spam detection algorithm is Figure 11, where x axis is frequency f in log scale and y axis is the conditional expression $D(f)$. According to Figure 11, it is easily for the spam detection algorithm to detect the spam, while the substring amplification cannot detect it.

4.4 Data from an Online Blog

The preceding two experiments used artificial data. We can estimate the lower bound, because we can change pa-

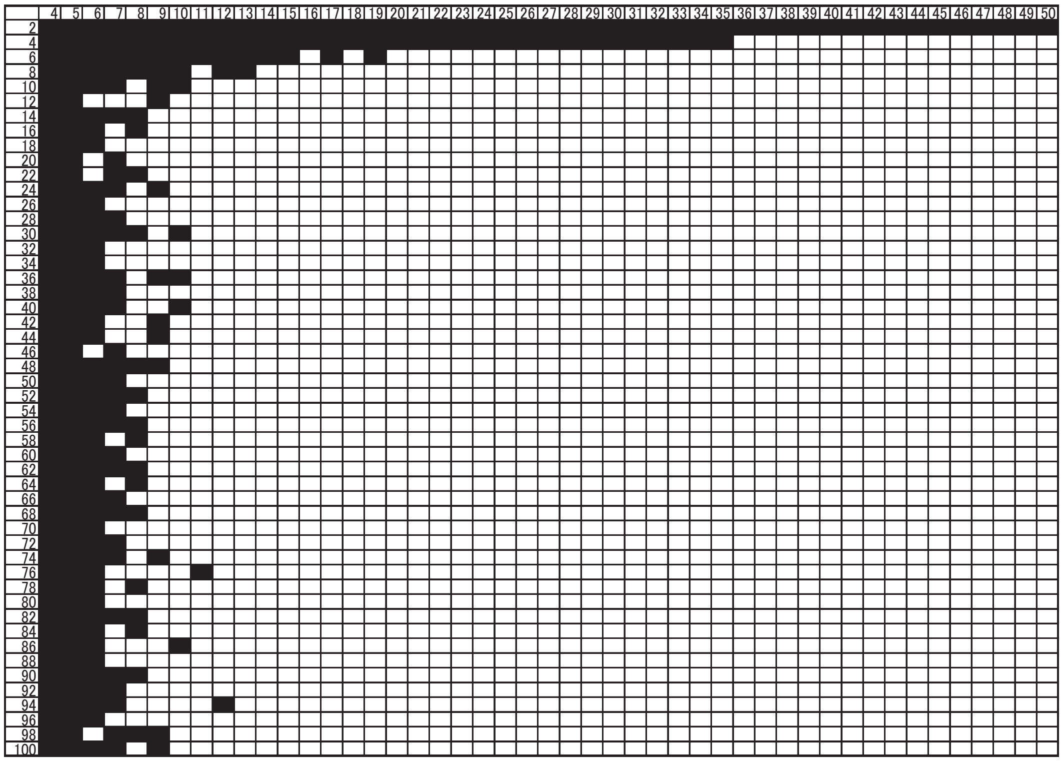


Figure 9: (i, j)-cell is a result of the sample which has j spams of length i.The black cells mean that the spam detection algorithm cannot detect spam, and white cells mean that the algorithm can detect spam

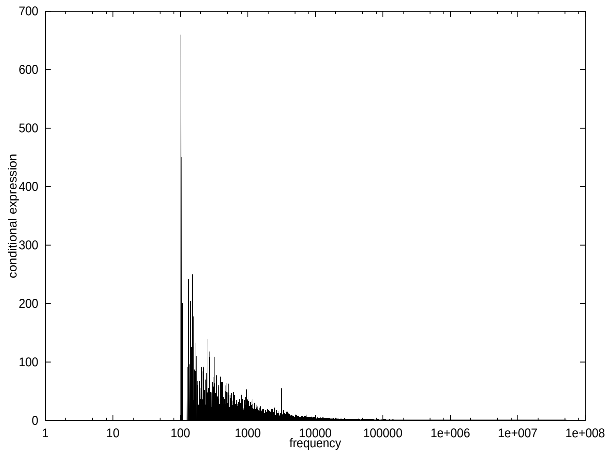


Figure 11: The conditional expression values $D(f)$ for the sample whose messages are 200000 of Date-Set2, where x-axis is frequency in log scale

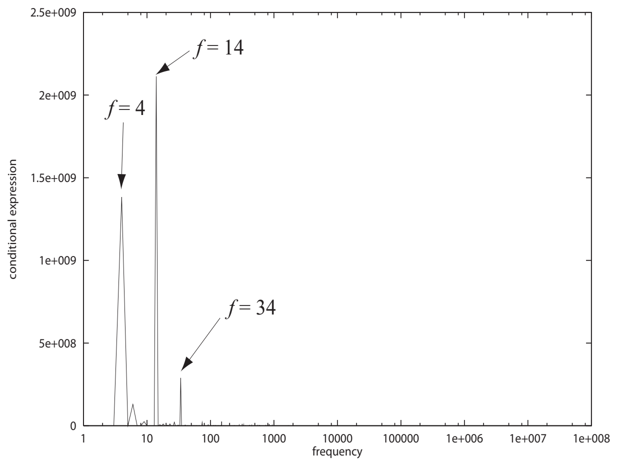


Figure 12: The conditional expression values $D(f)$ for the sample from "Arianna's Blog", where x-axis is frequency in log scale

rameters, such as the length, the number of spam messages, etc. In this section, we examine whether the spam detection algorithm can detect real spams among real messages from an online blog.

We use a sample collected from "Arianna's blog[5]". One file corresponds to one page which includes blog lists, blog contents, coments, or/and track back. HTML tags are removed in these files in order not to lead the tags to compose spikes. We collected 27,190 files whose total size is 151.09MB.

We do not know what kinds of spams are included, or moreover whether this sample includes spams or not. Fortunately, we found some spams. Table 3 is the result sorted by conditional expression $D(f)$. In Figure 12, x-axis is the frequency and in log scale, and y-axis is the conditional expression $D(f)$.

[5]`http://www.ariannaonline.com/blog/index.php`

Table 3: This table is the result of spam ditection algorithm for "Arianna's blog", whose rank is sorted by the value $D(f)$.

rank	f	V(f)	T(f)	D(f)
1	14	2129126914	29807776796	2113140742
2	4	1452192568	5808770272	1383153658
3	34	292089513	9931043442	289234115
4	6	163948953	983693718	129228621
5	150	25232763	3784914450	25203203
6	9	69918089	629262801	24948258
7	27	26797113	723522051	21702500
8	74	20290784	1501518016	20141172
9	316	11312892	3574873872	11310990
10	23	15330324	352597452	10934478
11	840	11435342	9605687280	10778713
12	18	17119809	308156562	10279396
13	16	15995164	255922624	5830974
14	92	5710183	525336836	5687658
15	334	4795475	1601688650	4794884
16	799	5218786	4169810014	4749966
17	284	4471440	1269888960	4469877
18	40	4855614	194224560	4378191
19	31	5971169	185106239	4256154
20	21	10835650	227548650	3644976

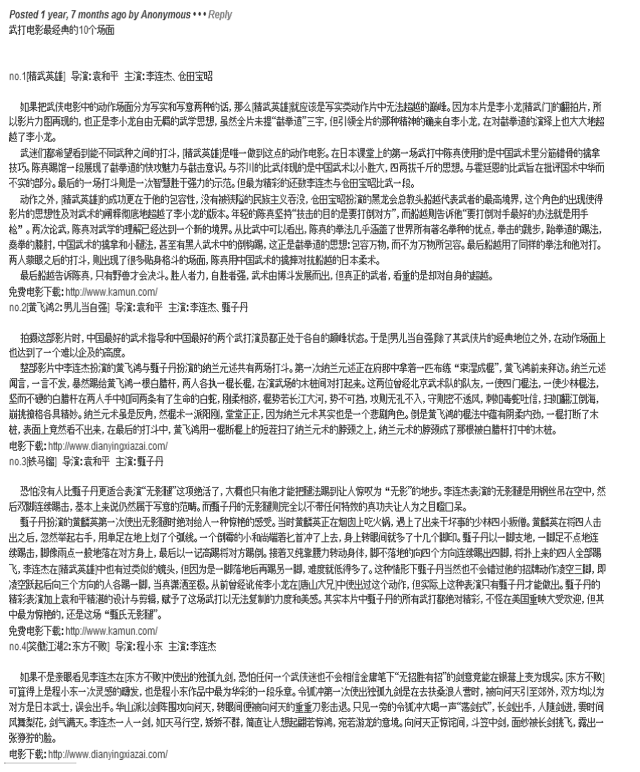
In Table 3 and Figure 12, the highest rank or spike is at $f = 14$. The longest string which appear fourteen times is a spam, in a general sense. The spam is the string of length 60,609 and the number of the spam is 3,768 words.

This spam detected by the spam detection algorithm includes several URLs which are not related with contents of this blog. The number of these URLs are aberration for one comment. Hence, these strings can be said spams in a general sense. The following strings are the part of the detected spam.

```
&#8226; @ &#8226; www &#8226; Reply
http://www.scorpion.my100megs.com
http://www.termites.ownsthis.com
http://bradpitt.freewebpage.org
http://www.quentin-tarantino.741.com
http://www.depeshe-mode.greatnow.com
http://www.mickeyrourke.esmartweb.com
http://www.crocodile.ownsthis.com
http://www.gitler.150m.com
http://www.titanic.741.com
http://www.chacknorrice.freewebpages.org
http://www.george-clooney.741.com
http://www.penelopecruz.batcave.net
```

...

```
http://milfseeker.com-top.net/terra.html
http://milfseeker.com-top.net/devon.html
http://teensforcash.com-top.net/FREE.html
http://teensforcash.com-top.net/KATHLEEN.html
http://teensforcash.com-top.net/ALEXIS.html
http://teensforcash.com-top.net/VIDEO.html
http://teensforcash.com-top.net/NICOLE.html
```



Posted 1 year, 7 months ago by Anonymous • • • Reply
武打电影最经典的10个场面

no.1[精武英雄] 导演:袁和平 主演:李连杰、仓田宝昭

如果把武侠电影中的动作场面分为写实和写意两种的话，那么[精武英雄]就应该是写实类动作片中无法超越的巅峰。因为本片是李小龙[精武门]的翻拍片，所以影片力图再现的，也正是李小龙自由无羁的武学思想，虽然全片未提"截拳道"三字，但引领全片的那种精神的确来自李小龙，在对截拳道的演绎上也大大地超越了李小龙。

武迷们都希望看到能不同武种之间的打斗，[精武英雄]是唯一做到这点的动作电影。在日本课堂上的第一场武打中陈真使用的是中国武术里分筋错骨的擒拿技巧。陈真踢馆一段展现了截拳道的快攻魅力与截击意识。与芥川的比武体现的是中国武术以小胜大，四两拔千斤的思想。与霍廷恩的比武旨在批评国术中华而不实的部分。最后的一场打斗则是一次智慧胜于强力的示范。但最为精彩的还数李连杰与仓田宝昭比武一段。

动作之外，[精武英雄]的成功更在于他的包容性，没有被狭隘的民族主义吞没，仓田宝昭扮演的黑龙会总教头船越代表武者的最高境界，这个角色的出现使得影片的思想性及对武术的阐释彻底地超越了李小龙的版本。年轻的陈真坚持"技击的目的是要打倒对方"，而船越则告诉他"要打倒对手最好的办法就是用手枪"。两次论武，陈真对武学的理解已经达到一个新的境界。从比武中可以看出，陈真的拳法几乎涵盖了世界所有著名拳种的优点，拳击的跳步，跆拳道的踢法，泰拳的膝肘，中国武术的擒拿和小腿法，甚至有黑人武术中的倒钩踢，这正是截拳道的思想：包容万物，而不为万物所包容。最后船越用了同样的拳法和他对打。两人蒙眼之后的打斗，则出现了很多贴身格斗的场面，陈真用中国武术的擒摔对抗船越的日本柔术。

最后船越告诉陈真，只有野兽才会决斗。胜人者力，自胜者强，武术由博斗发展而出，但真正的武者，看重的是却对自身的超越。

免费电影下载: http://www.kamun.com/
no.2[黄飞鸿2:男儿当自强] 导演:袁和平 主演:李连杰、甄子丹

拍摄这部影片时，中国最好的武术指导和中国最好的两个武打演员都正处于各自的巅峰状态。于是[男儿当自强]除了其武侠片的经典地位之外，在动作场面上也达到了一个难以企及的高度。

整部影片中李连杰扮演的黄飞鸿与甄子丹扮演的纳兰元述共有两场打斗。第一次纳兰元述正在府邸中拿着一匹布练"束湿成棍"，黄飞鸿前来拜访。纳兰元述闻言，一言不发，暴然踢给黄飞鸿一根白腊杆，两人各执一棍长棍，在演武场的木桩间对打起来。这两位曾经北京武术队的队友，一使四门棍法，一使少林棍法，坚而不硬的白腊杆在两人手中如同两条有了生命的白蛇，刚柔相济，棍势若长江大河，势不可挡，攻则无孔不入，守则密不透风，刺如毒蛇吐信，扫如翻江倒海，崩挑撩格各具精妙。纳兰元术虽是反角，然棍术一派阳刚，堂堂正正，因为纳兰元术其实也是一个悲剧角色。倒是黄飞鸿的棍法中蕴有阴柔内劲，一棍打断了木桩，表面上竟然看不出来，在最后的打斗中，黄飞鸿用一棍断棍上的短茬扫了纳兰元术的脖颈之上，纳兰元术的脖颈成了那根被白腊杆打中的木桩。

电影下载: http://www.dianyingxiazai.com/
no.3[铁马镏] 导演:袁和平 主演:甄子丹

恐怕没有人比甄子丹更适合表演"无影腿"这项绝活了，大概也只有他才能把腿法踢到让人惊叹为"无影"的地步。李连杰表演的无影腿是用钢丝吊在空中，然后双脚连续踢击，基本上来说仍然属于写意的范畴。而甄子丹的无影腿则完全以不带任何特效的真功夫让人为之目瞪口呆。

甄子丹扮演的黄麟英第一次使出无影腿时绝对给人一种惊艳的感受。当时黄麟英正在烟囱上吃火锅，遇上了出来干坏事的少林四小叛僧。黄麟英在将四人击出之后，忽然举起右手，用单足在地上划了个弧线。一个倒霉的小和尚端着匕首冲了上去，身上转眼间就多了十几个脚印。甄子丹以一脚支地，一脚足不点地连续踢击，脚像雨点一般地落在对方身上，最后以一记高踢将对方踢倒。接着又纯靠腰力转动身体，脚不落地的向四个方向连续踢出四脚，将扑上来的四人全部踢飞，李连杰在[精武英雄]中也有过类似的镜头，但因为是一脚落地后再踢另一脚，难度就低得多了。这种情形下甄子丹当然也不会错过他的招牌动作凌空三脚，即凌空跃起后向三个方向的人各踢一脚，当真潇洒至极。从前曾经讹传李小龙在[唐山大兄]中使出过这个动作，但实际上这种表演只有甄子丹才能做出。甄子丹的精彩表演加上袁和平精湛的设计与剪辑，赋予了这场武打以无法复制的力度和美感。其实本片中甄子丹的所有武打都绝对精彩，不怪在美国重映大受欢迎，但其中最为惊艳的，还是这场"甄氏无影腿"。

免费电影下载: http://www.kamun.com/
no.4[笑傲江湖2:东方不败] 导演:程小东 主演:李连杰

如果不是亲眼看见李连杰在[东方不败]中使出的独孤九剑，恐怕任何一个武侠迷也不会相信金庸笔下"无招胜有招"的剑意竟能在银幕上变为现实。[东方不败]可算得上是程小东一次灵感的礴发，也是程小东作品中最为华彩的一段乐章。令狐冲第一次使出独孤九剑是在去扶桑浪人营时，被向问天引至郊外，双方均以为对方是日本武士，误会出手。华山派以剑阵围攻向问天，转眼间便被向问天的重重刀影击退。只见一旁的令狐冲大喝一声"荡剑式"，长剑出手，人随剑进，霎时间风舞梨花，剑气满天。李连杰一人一剑，如天马行空，矫矫不群，简直让人想起翩若惊鸿，宛若游龙的意境。向问天正惊诧间，斗笠中剑，面纱被长剑挑飞，露出一张狰狞的脸。

电影下载: http://www.dianyingxiazai.com/

Figure 13: The spam detection algorithm detects the spam written by Chinese in Blog written by English. The length of this spam is 5,494, and the spam appears 4 times.

```
http://teensforcash.com-top.net/RANDI.html
http://teensforcash.com-top.net/MEGAN.html
http://teensforcash.com-top.net/KIMMIE.html
http://teensforcash.com-top.net/BIANCA.html
&bull; Trackback URL: http://ariannaonline.com/blog/
bblog/trackback.php?tbpost=2
```

Next, we remove this spam from dataset, and recount. When we repeat this operation, we detect other spam(Figure13). This spam is written by Chinese, and the length is 5,494.

5. CONCLUSION

We developed a spam detection algorithm whose key ideas are the Zipf's law and the vocabulary size. In our experiments using artificial data, the algorithm found blog spams whose length were greater than 10. So we can say that the algorithm finds blog spams practically. Furthermore, we showed that this algorithm can detect spams in real blog samples.

This algorithm judges a blog entry as a spam by the conditional expression $D(f)$, and the detected spam is removed from data set. The algorithm detects several kinds of spams because this algorithm repeats this operation. Thus there remains the problem how many repeat this operation does.

If a spammer copies a whole blog entry many times, the spam detection problem became much easier, because it is enough to just sort all blog entries alphabetically and then count the number of the same entries. However, there exist modified versions of one spam, such as the spam found in

Section 4. Therefore, we cannot conclude that counting unit is the whole message. In this situation, it is difficult for existing methods to detect spams.

The proposed method can detect some spams and templates. Thus, finally we must check whether found substrings are spams or templates. We expect that the frequency of spams is different from that of templates, because templates appear almost in data set but spams appear a few. We, however, do not know the difference clearly. We will detect all spams smartly by solving this.

6. REFERENCES

[1] R. Albert, H. Jeong, and A.-L. Barabási. Diameter of the World Wide Web. *Nature*, 401:130–131, 1999.

[2] R. Baeza-Yates, C. Castillo, and V. López. Pagerank Increase under Different Collusion Topologies. In *Proceedings of the First International Workshop on Adversarial Information Retrieval on the Web (AIRWeb)*, 2005. `http://airweb.cse.lehigh.edu/2005/`.

[3] A. A. Benczúr, K. Csalogány, T. Sarlós, and M. Uher. SpamRank – Fully Automatic Link Spam Detection. In *Proceedings of the First International Workshop on Adversarial Information Retrieval on the Web (AIRWeb)*, 2005. `http://airweb.cse.lehigh.edu/2005/`.

[4] S. Chhabra, W. S. Yerazunis, and C. Siefkes. Spam Filtering using a Markov Random Field Model with Variable Weighting Schemas. In *Proceedings of the 4th IEEE International Conference on Data Mining*, pages 347–350, November 2004.

[5] CNETNEWS.COM. Google Aims to Outsmart Search Tricksters, January 2005. `http://news.com.com/2100-1024_3-5540740.html`.

[6] CNETNEWS.COM. Tempted by Blogs, Spam Becomes 'Splog', October 2005. `http://news.com.com/2100-1032_3-5903409.html`.

[7] Z. Duan, Y. Dong, and K. Gopalan. DiffMail: A Differentiated Message Delivery Architecture to Control Spam. `http://www.cs.fsu.edu/research/reports/TR-041025.pdf`.

[8] D. Gusfield. *Algorithms on Strings, Trees, and Sequences : Computer Science and Computational Biology*. Cambridge University Press, 1997.

[9] Z. Gyöngyi and H. Garcia-Molina. Web Spam Taxonomy. In *Proceedings of the First International Workshop on Adversarial Information Retrieval on the Web (AIRWeb)*, 2005. `http://airweb.cse.lehigh.edu/2005/`.

[10] D. Ikeda. *Autoschediastic Text Mining Algorithms*. PhD thesis, Graduate School of Information Science and Electrical Engineering, Kyushu University, March 2004.

[11] D. Ikeda and Y. Yamada. Gathering Text Files Generated from Templates. In *Proceedings of Workshop on Information Integration on the Web (IIWeb-04)*, pages 21–26, August 2004. `http://cips.eas.asu.edu/iiweb.htm`.

[12] T. Kasai, G. Lee, H. Arimura, S. Arikawa, and K. Park. Linear-time Longest-Common-Prefix Computation in Suffix Arrays and Its Applications. In *Proceeding of the 12th Annual Symposium on Combinatorial Pattern Matching*, Lecture Notes in Computer Science 2089, pages 181–192. Springer-Verlag, 2001.

[13] D. K. Kim, J. S. Sim, H. Park, and K. Park. Linear-time construction of suffix arrays. In *Proceeding of 14th Combinatorial Pattern Matching*, Lecture Notes in Computer Science 2676, pages 186–199, 2003.

[14] G. Manzini and P. Ferragina. Engineering a Lightweight Suffix Array Construction Algorithm. *Algorithmica*, 40(1):33–50, 2004.

[15] G. Mishne, D. Carmel, and R. Lempel. Blocking Blog Spam with Language Model Disagreement. In *Proceedings of the First International Workshop on Adversarial Information Retrieval on the Web (AIRWeb)*, 2005. `http://airweb.cse.lehigh.edu/2005/`.

[16] M. Nagao, S. Sato, S. Kurohashi, and T. Tsunoda. *Natural Language Processing*. IWANAMI KOZA Software Science 15. Iwanami Shoten, April 1996. (in Japanese).

[17] M. Sahami, S. Dumais, D. Heckerman, and E. Horvitz. A Bayesian Approach to Filtering Junk E-mail. *AAAI Workshop on Learning for Text Categorization*, July 1998. `ftp://ftp.research.microsoft.com/pub/ejh/junkfilter.pdf`.

[18] B. Wu and B. D. Davison. Identifying Link Farm Spam Pages. *In Proceedings of the 14th International World Wide Web Conference*, 2005.

[19] K. Yoshida, F. Adachi, T. Washio, H. Motoda, and et al. Memory Management of Density-Based Spam Detector. *Symposium on Applications and the Internet (SAINT'05)*, pages 370–376, 2005.

[20] G. K. Zipf. *The Psycho-Biology of Language: An Introduction to Dynamic Philology*. Houghton Mifflin, 1935.

[21] G. K. Zipf. *Human Behavior and the Principle of Least Effort*. Addison-Wesley, 1949.