Eliminating Useless Parts in Semi-structured Documents Using Alternation Counts

Ikeda, Daisuke
Computing and Communications Center, Kyushu University

Yamada, Yasuhiro
Graduate School of Information Science and Electrical Engineering, Kyushu University

Hirokawa, Sachio
Computing and Communications Center, Kyushu University

出版情報：Lecture Notes in Computer Science. 2226, pp.113-127, 2001-11. Springer
バージョン：

Eliminating Useless Parts in Semi-structured Documents using Alternation Counts

Daisuke Ikeda[1], Yasuhiro Yamada[2], and Sachio Hirokawa[1]

[1] Computing and Communications Center,
Kyushu University, Fukuoka 812-8581, Japan
{daisuke,hirokawa}@cc.kyushu-u.ac.jp
[2] Graduate School of Information Science and Electrical Engineering,
Kyushu University, Fukuoka 812-8581, Japan
yshiro@matu.cc.kyushu-u.ac.jp

Abstract. We propose a preprocessing method for Web mining which, given semi-structured documents with the same structure and style, distinguishes useless parts and non-useless parts in each document without any knowledge on the documents. It is based on a simple idea that any n-gram is useless if it appears frequently. To decide an appropriate pair of length n and frequency a, we introduce a new statistic measure *alternation count*. It is the number of alternations between useless parts and non-useless parts. Given news articles written in English or Japanese with some non-articles, the algorithm eliminates frequent n-grams used for the structure and style of articles and extracts the news contents and headlines with more than 97% accuracy if articles are collected from the same site. Even if input articles are collected from different sites, the algorithm extracts contents of articles from these sites with at least 95% accuracy. Thus, the algorithm does not depend on the language, is robust for noises, and is applicable to multiple formats.

1 Introduction

Data mining is a research field to develop tools that find useful knowledge from databases [3, 4, 14]. In this field, databases is assumed to have explicit and static structures. On the other hand, resources on the WWW do not have such structures. Web mining is a field of mining from such resources and text mining is mining from unstructured or semi-structured documents. We consider Web or text mining from semi-structured documents. A semi-structured document have tree structures, such as HTML/XML files, BiBTeX files, etc [1].

Since resources on the Web are widely distributed and heterogeneous, it is important to collect documents and clean them. When we collect a large amount of documents, we use hyperlinks for efficiency. For example, search engines provide hyperlinks. Since a hyperlink is not created by the collecter, we can not assume that collected documents are well cleaned. Some of them are written in different languages and are far from the desired topic. Thus, Web mining algorithms and preprocessors should be robust for noise and should be applicable to any natural and markup languages.

Usual mining algorithms assume that collected documents are written in the same language and preprocess them with some knowledge, such as the grammar of HTML to remove tags, stop word lists [10], stemming technique [15], morpheme analysis, etc. They depends on natural and markup languages. Some algorithms require additional input documents as background knowledge. In addition to an input set of documents, the algorithm in [5] requires another set of documents in order to remove substrings highly common to both sets.

In this paper, we present an algorithm that cleans collected documents without any knowledge on them. From input documents, this algorithm finds a set of frequent n-grams. Using this set, we can eliminate tags or directives, and stereotyped expressions in the documents because they are common to the collected documents and so useless. Eliminating or finding *useless* parts contrasts with usual mining algorithms which find *useful* knowledge such as association rules [3], association patterns [4], word association patterns [5], and episode rules [14].

An input for our algorithm is a set of semi-structured documents which contain the same structure and style such as static Web pages in the same site or dynamic pages generated with search facility. Since the number of Web sites providing search facilities is increasing [7], the number of Web pages applicable to our algorithm is large. In such pages, there exist frequent substrings, such as the name of the site, navigation and advertisement links, etc. Moreover, there exist common tags or directives which structure texts because they have the same structure and style. If we see such pages, we usually put our mind on the variable part and ignore invariable parts. In this sense, substrings to describe the same structure and style in such pages are useless.

We treat a document as just a string and define any frequent n-grams are useless. An n-gram is just a string with length n, so that our algorithm does not depend on natural and markup languages. Once an appropriate pair (n, a), which is called a *cut point*, of a length n and frequency a is decided, we divide each of documents into two parts using the pair as follows: if the frequency of an n-gram is in the top a percent of the frequencies of all n-grams, then the n-gram is useless.

To decide an appropriate pair (n, a), we introduce a new statistic measure *alternation count*. It is the number of changes between useless parts and non-useless parts in a document. For a set D of documents, alternation count of D is the sum of all alternation counts. An alternation count shows how many times useless (or non-useless) parts appear in documents. A large alternation count splits a document into too small pieces. In this case, structures of each document are destroyed. Therefore, our algorithm searches cut points from $(2, 1)$ while the alternation count of the current cut point is decreasing. The algorithm stops if the alternation count becomes to be greater than the current one when it increases n or a by one.

We define an *optimal* cut point (n, a) which attains a locally minimum alternation count and develop an algorithm that finds an optimal cut point. It runs in $O(n^2N + nN \log N)$ time, where N is the total length of input and n is the

length of an optimal cut point. Experimentally, n is less than 30 and $n \ll N$, so the time complexity is approximately $O(N \log N)$.

We present experimental results using news articles as input for the algorithm. The articles are written in English or Japanese. For articles collected from the same site, the algorithm detects the tag regions of HTML files and highly common expressions in the articles as useless. It extracts the news contents as non-useless parts with more than 97% accuracy even if non-articles are contained as noise data. Therefore, the algorithm does not depend on the language and is robust for noises.

To evaluate experimental results, we manually define non-useless parts of articles for each data set. Comparing the manually defined division with the division by the algorithm, the accuracy is defined to be the number of letters categorized to the same part (useless or non-useless) to the total length of the input.

We also evaluate the accuracy for documents collected from different sites with any combination of English and Japanese sites. The algorithm extracts contents of articles from these sites with at least 95% accuracy. Since the algorithm simply counts n-gram frequencies, a extremely small set of articles would be ignored in a combination with large data sets. However, some expriments show that the algorithm detects the contents of articles in such a small set as well as those in a large set.

This paper is organized as follows. In the next section, basic notations are given and then the key notion, the alternation count, is introduced. In Section 3, we present an algorithm that divides a document of given documents into useless and non-useless parts. Complexity required by the algorithm is presented in Section 3.1. Experimental results are shown in Section 4.

2 Alternation Count and Optimal Cut Point

2.1 Preliminaries

The set Σ is a finite alphabet. Let $x = a_1 \cdots a_n$ ($a_i \in \Sigma$ for each i) be a string over Σ. We denote the *length* of x by $|x|$. An *n-gram* is a string whose length is n. For an integer $1 \leq i \leq |x|$, we denote by $x[i]$ the ith letter of x. Let x and y be two strings. The concatenation of x and y is denoted by $x \cdot y$ or simply by xy. We denote $x = y$ if $|x| = |y|$ and $x[i] = y[i]$ for each $1 \leq i \leq |x|$.

For a string x, if there exist strings $u, v, w \in \Sigma^*$ such that $x = uvw$, we say that v is a *substring* of x. An *occurrence* of v in x is a positive integer i such that $x[i] \cdots x[i + |v| - 1] = v$. Using the occurrence and the length of v, v is also denoted by $x[i..i + |v| - 1]$.

If $\Sigma = \{0, 1\}$, then a string over Σ is called a *binary* string. For $i \in \Sigma$ and $x \in \Sigma^*$, $[x]_i$ denotes the number of i's in x. For two binary strings x and y with the same length, bitwise "and" and "exclusive-or" operations defined as follows. $x\&y$ is a binary string with length $|x|$ such that $x\&y[i] = 1$ if $x[i] = y[i] = 1$ and $x\&y[i] = 0$ otherwise. $x \hat{} y$ is also a binary string with length $|x|$ such that

$x\hat{}y[i] = 1$ if $x[i] \neq y[i]$ and $x\hat{}y[i] = 0$ otherwise. For example, if $x = 01101$ and $y = 11100$, then $x\&y = 01100$, $x\hat{}y = 10001$, $[x\&y]_0 = 3$, and $[x\hat{}y]_1 = 2$.

Let x be a string (not limited to be binary) and $W = \{v_1, \ldots, v_n\}$ be a set of substrings of x. A *range string* of W on x, denoted by $r_x(W)$, is a binary string with length $|x|$ such that $r_x(W)[j] = 0$ if $i \leq j \leq i + |v_k| - 1$ for some occurrence i of v_k $(1 \leq k \leq n)$ and $r_x(W)[j] = 1$ otherwise. A successive 0's on the range string shows intervals on x covered by the substrings in W. For example, let $x = accbaacbc$ and $W = \{cb, ba\}$. Then $r_x(W) = 110001001$.

2.2 Alternation Count

In this section, we introduce the key notion *alternation count*. We consider semi-structured documents with the same structure and style such as static pages of one site and dynamic pages created by a search facility. In such documents, there exists frequent substrings for the structure and style and many users are not interested in them. Therefore, we define useless parts of documents as follows.

Definition 1. *Let D be a set of strings. Then, a substring of a string in D is said to be* useless *when it appears frequently in D.*

We treat a semi-structured document as just a string. Note that we do not define "importance", "siginificance", or "usefulness" like other researchs on text and Web mining.

Next, we consider how many times a substring appears we can say that it does "frequently". The measure for it is new notion alternation count. It is, given a string and a set of substrings of the string, the number of changes from a part of the string covered by given substrings to the other part, and vice versa.

Definition 2. *Let x be a string and W be a set of substrings of x. Then, the* alternation count *of W on x, which is denoted by $A_x(W)$, is the number of boundaries between different value's (0 and 1) on the range string $r_x(W)$.*

Example 1. Let $x = accbaacbc$ and $W = \{cb, ba\}$. Then $A_x(W) = 4$ because $x = ac\underline{cba}a\underline{cb}c$ (a part of underlined letters is cb or ba) and $r_x(W) = 110001001$.

The above definition is easily extended to a set of strings instead of a single string x. The alternation count of W on a set D of strings is the sum of alternation counts of $x \in D$ and denoted by $A_D(W)$.

2.3 Optimal Cut Point

Our algorithm is required to receive a set of semi-structured documents and divide them into two parts of substrings — useless parts and non-useless parts.

Since we treat a semi-structured document as just a string, an input for our algorithm is a set $D = \{x_1, x_2, \ldots, x_n\}$ of strings. To express useless or non-useless parts, we use a set of substrings of $x_i \in D$. Thus, our algorithm is required to receive D and decide W such that consecutive 0's on $r_{x_i}(W)$ $(i = 1, 2, \ldots, n)$ cover substrings on x_i for the structure and style.

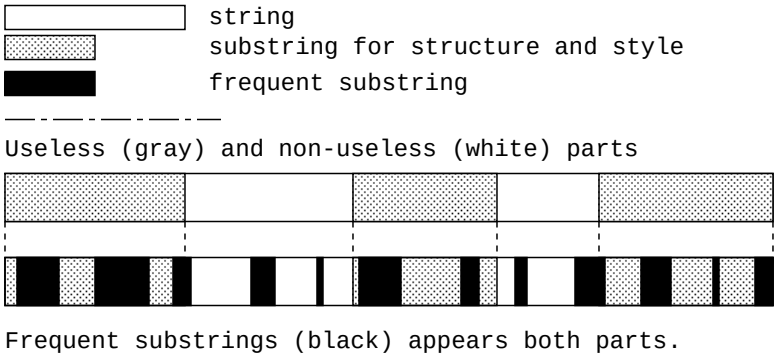

Fig. 1. Two strings are the same. The above one shows that where are substrings for the structure and style. The other one shows that the frequent substrings (black parts) fails to cover gray parts

The most simple method to find frequent substrings of D is to enumerate all substrings of D and decide a boundary between frequent substrings and non-frequent ones according to some measure. However, W constructed by this method may contain short substrings and they appears in non-useless parts as well as useless parts. And, the method requires a large time complexity because there exist $O(N^2)$ substrings for a string with length N.

Instead of this, we make the algorithm to decide the appropriate length of substring as well as the appropriate frequency. In other word, we use n-grams instead of substrings with any length. In our algorithm, the frequency is expressed by a percentage. A pair (n, a) of a length and a frequency decides W such that W is the set of the top a percent frequent n-grams in D. In the sequel, we denote $A_D(W)$ by $A_D(n, a)$. The pair (n, a) is called a *cut point* of D. An n-gram is said to be *frequent* on a cut point (n, a) if the n-gram in W decided by (n, a).

Since an appropriate cut point depends on input documents, we make the algorithm to it automatically. First, we consider what is an appropriate cut point. Two strings in Fig. 1 show the same string. The above string shows that substrings for the structure and style are colored with gray. An appropriate (n, a) covers gray parts with frequent n-grams. The below string shows that frequent n-grams appear in both gray and white parts. In this case, the alternation count is larger than the ideal alternation count and substrings for the structure and style are destroyed.

Since short substrings seems not to construct structures and styles, they appear everywhere. Therefore, an alternation count may be large if n is too small. An alternation count also may be large if a is smaller than the ideal one since increasing a connects separate frequent n-grams. If n or a is greater than the corresponding ideal one, the alternation count also become large. Thus, an

appropriate cut point if both n and a are enough large and the pair (n, a) attains a locally minimum alternation count.

A *path* is a sequence of cut points $(n_1, a_1), (n_2, a_2), \ldots, (n_k, a_k)$ such that (1) either $n_{i+1} = n_i + 1$ or $a_{i+1} = a_i + 1$ for each $i = 1, 2, \ldots, k-1$, (2) $A_D(n_i, a_i) > A_D(n_{i+1}, a_{i+1})$ for each $i = 1, 2, \ldots, k-1$, and (3) $A_D(n_k, a_k) < A_D(n_k + 1, a_k)$ and $A_D(n_k, a_k) < A_D(n_k, a_k + 1)$. A cut point (n, a) is *optimal* if there exists a path from $(2, 1)$ to (n, a). Note that, there exist some optimal cut points.

The trivial initial cut point is $(1, 1)$. However, 1-gram is too short to describe the structure and style of documents. Thus, we define the initial cut point is $(2, 1)$.

Using above notions, we define that eliminating useless parts is, given a set D of strings, to find an optimal cut point of D.

3 Algorithm

In this section, we describe `FindOptimal` that finds an optimal cut point (n, a) (see Fig. 3). From the initial cut point $(2, 1)$, the algorithm compares alternation counts on the next two cut points with the currnet one. It stops when alternation counts on both next two cut points are greater than the current one.

The algorithm does not remove any tags or directives of semi-structured documents. It only modifies documents according to the following conventional preprocessing rules: tabs and newlines are treated as a space, and consecutive spaces are compressed into one space. An input for the algorithm is a set of strings preprocessed according to the above rules.

`FindOptimal` uses two subroutines `alternation` (see Fig. 2) and `countsort`. The subroutine `countsort` receives an integer n and a set D of strings, then counts all n-grams in D and sorts them by the number of their occurrences. The subroutine keeps the n-grams and the numbers of their occurrences in a hash table. It returns an array of the counted substrings sorted in the decreasing order.

The subroutine `alternation` receives a set D of strings, an array O of strings, a length n, and a percentage a. The variable W used in `alternation` keeps the first $a/100$ strings in the sorted array O which is an output of `countsort`. Using W, the subroutine construct a range string r and then counts the boundaries, which is the alternation count on (n, a).

The main algorithm `FindOptimal` (see Fig. 3) receives a set D of strings. It counts the alternation count on the current cut point and also counts alternation counts on next two candidates, $(n, a+1)$ and $(n+1, a)$. After comparison alternation counts on these three cut points, it decides the next cut point. If both $A_D(n, a+1)$ and $A_D(n+1, a)$ are small than $A_D(n, a)$, the algorithm selects the cut point providing a smaller alternation count. If there is no next cut point providing a smaller alternation count than the current one, then it returns the currnet cut point as an optimal cut point.

```
function alternation (var D: set of strings;
  O: array of strings, n: integer ; a: integer): integer;
  var
    i: integer;
    s, x: string;
    W: hash table;
    r: array [1..|x|] of integers;
  begin
    x := string concatenated all strings in D;
    for i := 1 to |x| do r[i] := 1;
    W := substrings
      from the first substring to the a/100-th substring in O;
    for i := 1 to |x| do begin
      s := x[i..i + n] ;
      if s ∈ W then
        r[i..i + n] := 0 ;
    end {for}
    count boundaries between r[i] = 0 and r[i] = 1;
    return the boundaries;
  end ;
```

Fig. 2. The subroutine returns the alternation count of D on (n, a)

3.1 Complexity

In this section, we discuss the time complexty required by `FindOptimal`. Let D be an input set of strings and N be the total length of D.

First, we estimate the complexity required by the subroutine `countsort`. It is required $O(1)$ time to add a new n-gram to a hash table or to check if a given n-gram is already stored in the hash table. Since the subroutine counts all substrings with the same length, there exist at most $O(N)$ n-grams. Therefore, `countsort` needs $O(N)$ time to construct the hash table and $O(N \log N)$ time to sort n-grams. Thus, `countsort` runs in $O(N \log N)$ time.

Next, we consider the subroutine `alternation`. Let (D, O, n, a) be an input for the subroutine. $O(N)$ time is required to construct a hash table W. In the last **for** loop, check if $s \in W$ requires $O(1)$ using the hash table W and writing 0 on $r[i..i + n]$ requires $O(n)$ time. Therefore, this loop is completed in $O(nN)$ time. Counting boundaries is done in $O(N)$ by scanning r from left to right. Thus, the subroutine runs in $O(nN)$ time.

Finally, we estimate the time complexity of `FindOptimal`. Let (n_f, a_f) be the final output of `FindOptimal`. `FindOptimal` calls `countsort` $n_f - 1$ times and `alternation` at most $3(n_f + a_f - 2) + 1$ times because it passes through $n_f + a_f - 2$ cut points from the initial cut point $(2, 1)$ to (n_f, a_f). Thus, the routine runs in the following time:

$$\begin{aligned} & O(n_f N \log N + (n_f + a_f) n_f N) \\ = & O(n_f N \log N + n_f^2 N + a_f n_f N) \end{aligned}$$

```
procedure FindOptimal (var D: set of strings);
  {FindOptimal finds an optimal cut point.}
  var n, a: integer;
  {n and a keep the current length and percentage, respectively.}
  var val_0, val_1, val_2: integer;
  var O_cur, O_next: array of strings;
  begin
    n := 2; a := 1 ; {initialize n and a.}
    O_cur := countsort(D, n);
    val_0 :=alternation(D, O_cur, n, a) ;
    {val_0 keeps the alternation count on the current (n, a).}
    while (n ≤ max{|d| | d ∈ D}) and (a < 100) do begin
      if countsort(D, n + 1) is not done then
         O_next := countsort(D, n + 1);
      val_1 :=alternation(D, O_cur, n, a + 1);
      val_2 :=alternation(D, O_next, n + 1, a);
      {val_1 and val_2 keep the alternation counts on next candidates.}
      if (val_0 ≤ val_1) and (val_0 ≤ val_2) then
        goto OUTPUT ;
      else if (val_0 > val_1) and ( val_1 < val_2) then begin
        a := a + 1 ;
        val_0 := val_1 ;
      end
      else if ((val_0 > val_2) and (val_2 ≤ val_1)) then begin
        n := n + 1 ;
        val_0 := val_2 ;
        O_cur := O_next ;
      end
    end ; {while}
    OUTPUT :
    report (n, a) ;
  end ;
```

Fig. 3. The main algorithm `FindOptimal` outputs an optimal cut point using two subroutines `countsort` and `alternation`

$$= O(n_f^2 N + n_f N \log N),$$

where a_f is a non-negative constant less than 100. Our experimental results show that n_f is less than 30 and $n \ll N$ (see Section 4). Thus, the time complexity is approximately $O(N \log N)$.

4 Experiments

We use news articles as input for `FindOptimal`. An article we use is written in English or Japanese. It is provided as an HTML file which has the headline and the body of it.

We have two types of experiments depending on the the number of sites from which we collect articles: articles from the same site (see Section 4.2) and articles from different sites (see Section 4.3).

4.1 Evaluation

To evaluate an outputed cut point, we utilize two approaches. We modify HTML files in which any frequent n-gram on the cut point is colored with gray like $ac\mathit{cba}a\mathit{cb}c$. A colored letter corresponds to 0 on the range string 110001001.

The other approache is to calculate accuracy, recall, and precision using a binary string called a *correct string*. We can conclude that `FindOptimal` outputs an appropriate cut point if a range string $r_x(n, a)$ is similar to the corresponding correct string.

Let D be an input for `FindOptimal` and x be a string concatenated all strings in D. A correct string c is a binary string with length $|x|$ such that, for $1 \leq i \leq |c|$, $c[i] \in \{0, 1\}$ is decided according to manually specified pairs of delimiters. Let (l, r) be a pair of delimiters and u be a substring of c. Then, $u = 11 \cdots 1$ if lur be a substring of x, $u = 00 \cdots 0$ otherwise. A substring surrounding with the pair is the headline or the body of an article in our experiments. The positions corresponding to the substring are filled with 1 in the correct string.

We define that the body and the headline of an article are not useless, extract manually left and right delimiters from each data set, and construct correct strings using pairs of delimiters. Then, using two binary string, the correct string c and the range string r, we define that accuracy is $[c \hat{} r]_0/|r|$, recall is $[c\&r]_1/[c]_1$, and precision is $[c\&r]_1/[r]_1$. The accuracy is the ratio of positions i such that $c[i] = r[i]$ to the total length of input documents.

4.2 Articles from the Same Site

In this section, two sets of documents are considered. One is a set of 76 articles obtained from "The Washington Post (`http://www.washingtonpost.com/`)". This set is denoted by *WPOST*. All pages linked from the URL are collected and then non-article pages are removed manually. Articles in any categories are included, and so do any other data sets in this paper. The total size of WPOST

<HTML><HEAD> <style type="text/css">□□□1370□□□ <META NAME="edition" CONTENT="M2"> <META NAME="document_name" CONTENT="A31243-2001Jan22"> <META NAME="source" CONTENT="Post"> <META NAME="section" CONTENT="DM"> <META NAME="page" CONTENT="E01 "> <META NAME="column" CONTENT=""> <META NAME="slug" CONTENT="BANK23"> <META NAME="timestamp" CONTENT="07:08 AM"> <META NAME="category" CONTENT="BIZ"> <META NAME="wordcount" CONTENT="0"> <META NAME="sourceNumber" CONTENT="6"> <!–plsfield:title–> <TITLE>McColl Shuts Books on an Era (washingtonpost.com)</TITLE> </HEAD>□□□15200□□□ <!–plsfield:headline–> <FONT FACE="Arial,Helvetica" SIZE="+1"><B>McColl Shuts Books on an Era</B></FONT> <!–plsfield:stop–>□□□2050□□□ <A HREF="/cgi-bin/gx.cgi/AppLogic+FTContentServer?pagename=wpni/email&articleid=A31243-2001Jan22&node=business"><B>E-Mail This Article</B></A>
</FONT></TD> □□□470□□□<A HREF="/ac2/wp-dyn/A31243-2001Jan22?language=printer"><B> Printer-Friendly Version</B></A>
</FONT></TD> <TD WIDTH="8" HEIGHT="1"><SPACER TYPE="block" WIDTH="8" HEIGHT="1"></TD></TR> <TR><TD COLSPAN="4" WIDTH="226" HEIGHT="8"><SPACER TYPE="block" WIDTH="226" HEIGHT="8"></TD></TR></TABLE> </TD></TR> <TR><TD WIDTH="228" HEIGHT="1"><SPACER TYPE="block" WIDTH="226" HEIGHT="1"></TD></TR></TABLE> </TD></TR></TABLE> <FONT SIZE="2"> <!–plsfield:byline–> <I>By Kathleen Day</I>
 <!–plsfield:credit–> Washington Post Staff Writer
 <!–plsfield:disp_date–> Tuesday, January 23, 2001; Page E01
 </FONT> </P> <!–plsfield:description–> <P><P></P> <P><P>The expected resignation tomorrow of Hugh McColl as head of Bank of America symbolically marks the end of an era in banking, where for two decades mergers have been an engine of growth and the idea that bigger is better has been gospel.</P> <P><P> His departure from the nation's largest consumer bank comes less than a year after his fierce crosstown competitor in Charlotte, Edward C.□□□4580□□□ "They have a mutual respect for each other," said Virginia Stone Mackin, spokeswoman for First Union, who until a few years ago worked at Bank of America.</P> <P><P> McColl was among the first to call Crutchfield when he was diagnosed with cancer, she said. And the two have been known to spent the evening talking after bumping into one another at parties or the country club.</P> <!–plsfield:end–> <P><CENTER> © 2001 The Washington Post Company </CENTER></P> <P><CENTER> <A HREF="/wp-dyn/business/A32068-2001Jan22.html"><img src="http://a188.g.akamaitech.net/f/188/920/1d/www.washingtonpost.com/wp-srv/images/arrow_previous.gif" width="14" height="12" border="0"> □□□8700□□□ <A HREF="http://www.washingtonpost.com/wp-srv/maps/mit_foto.map"><IMG SRC="http://a188.g.akamaitech.net/f/188/920/1d/www.washingtonpost.com/wp-srv/images/channelnav_news.gif" WIDTH="760" HEIGHT="16" BORDER="0" ALT="channel navigation" ISMAP="true"></A>
</TD> </TR><TR> <TD></TD> <TD><IMG SRC="http://a188.g.akamaitech.net/f/188/920/1d/www.washingtonpost.com/wp-srv/globalnav/images/spacer.gif" WIDTH="428" HEIGHT="1" BORDER="0" ALT=" "></TD> <TD></TD> <TD></TD> </TR></FORM></TABLE> <FONT SIZE="-2">
</FONT> </TD></TR></TABLE> </BODY></HTML>

Fig. 4. An HTML file of WPOST where frequent n-grams on $(24, 10)$ are colored with gray

<html> <head> <title> Yomiuri On-Line/□□□5250□□□ 小泉首相、今国会への補正予算案提出を否定
 <img src="/g/d.gif" width="1" height="5">
 <img src="/g/db.gif" width="465" height="1">
<img src="/g/d.gif" width="1" height="15">
 <!– photo start –> <!– NO PHOTO –> <!– photo end –> <!– honbun start –> 小泉首相は一日、首相官邸で記者団に対し、景気対策として二〇〇一年度補正予算案を今国会に提出する可能性について、「考えていない」と否定した。   首相はこれまで、従来の公共事業中心の景気対策には否定的な立場をとっている。また、財政再建に向け、国債発行額を毎年三十兆円以下に抑える考えも示しており、補正予算編成への消極姿勢も、こうした基本方針を反映したものだ。   さらに、政府・与党が緊急経済対策に盛り込んだ、財政出動を伴う可能性のある「銀行保有株式取得機構」にも、首相は「早急につくるのではなく、もう少し専門家に意見を聞き、より充実したものにすべきだ」と、慎重に内容を検討する意向を示している。 (5 月 1 日 21: 19)
 <!– honbun end –> <img src="/gif6/arw.gif" width="11" height="11">□□□5800□□□ <LAYER SRC="/srcfiles/specials.htm" VISIBILITY=hidden ONLOAD="moveToAbsolute(specials.pageX, specials.pageY); visibility=true;"></LAYER> </body> </html>

Fig. 5. An HTML file of YOMIURI where frequent n-grams on $(27, 10)$ are colored with gray

is about 3.2M Bytes (average 42K Bytes), the minimum size of the articles is 31K Bytes, and the maximum size is 65K Bytes.

Given WPOST, `FindOptimal` outputs $(24, 10)$ as an optimal cut point. Fig. 4 is an HTML file in WPOST where frequent n-grams on $(24, 10)$ are colored with gray. Some letters are omitted because the file is too large. A number surrounded by three boxes □, such as □□□5250□□□, denotes the approximative number of omitted letters. The color of omitted letters is the same as the color of the boxes and the number. In Fig. 4, the longest black substring completely equals to the body of the article except for the last period. Short black strings are substrings of the headline, which appear twice, or a substring of the file name in which this article is contained. A file name is unique in this set, so that a part of the file name is remains not to be colored.

The accuracy, recall, and precision on WPOST is 0.975, 0.939, and 0.872, respectively. The precision is relatively lower because unique substrings are included in the header of an HTML file such as a part of the file name.

The other set consists of articles written in Japanese. They are collected from "Yomiuri On-Line (`http://www.yomiuri.co.jp/`)". This is constructed by collecting all linked pages from all categories. For example, economic articles are collected from `http://www.yomiuri.co.jp/02/index.htm`. This set is denoted by *YOMIURI*. The number of the articles in YOMIURI is 198, the total size of YOMIURI is about 2.7M Bytes (average 13.4K Bytes), the minimum size of the articles is 297 Bytes, and the maximum size is 39K Bytes.

YOMIURI includes non-article files as noise data. The shortest file is not an article file. Moreover, files whose size are smaller than 12K Bytes are top pages of categories. Such files have only hyperlinks to articles. The number of noises in YOMIURI is 14 and its size is about 36K Bytes.

Given YOMIURI, `FindOptimal` outputs $(8, 10)$ as an optimal cut point. The accuracy, recall, and precision on YOMIURI is 0.992, 0.808, and 0.991, respectively.

Fig. 5 is a modified HTML file. Black parts are parts of the headline and the body of the article. The headline is at the second line and the content starts after "`<!-- honbun start -->`" which means the start of the body. Short colored substrings in the body are "<p>" tags with the beginnings or ends of sentences. These substrings decrease the recall. Ends of sentences appear frequently in a Japanese sentence and beginnings are frequent expressions in articles such as "prime minister (Koizumi)".

4.3 Articles from Different Sites

We use "Los Angeles Times (`http://www.latimes.com/`)" and "The Sankei Shimbun (`http://www.sankei.co.jp/`)" in addition to sites described in Section 4.2. Two sets of articles collected from the URLs are denoted by *LATIMES* and *SANKEI*, respectively. Articles in LATIMES and SANKEI are written in English and Japanese, respectively.

In this section, the following data sets are considered. Any combination of English and Japanese sites is included.

WP-LA articles from WPOST (30 articles 2.36M Bytes) and LATIMES (40 articles 2.44M Bytes)

SA-YO articles from SANKEI (23 articles 86K Bytes) and YOMIURI (198 articles 2.7M Bytes)

SA-LA articles from SANKEI (23 articles 86K Bytes) and LATIMES (150 articles 4.5M Bytes)

Note that the size of data set SANKEI is too small.

Table 1 shows outputs of `FindOptimal` and evaluation values. A cell $x(y, z)$ of an evaluation value shows the total value x on all articles and two evaluation values y and z on each site articles. The total value is not the average of the corresponding two single sites. It is calculated by just counting letters of all articles from both sites. That is, if two evaluation values of single site are x_1/y_1 and x_2/y_2, then the total evaluation is $(x_1 + x_2)/(y_1 + y_2)$. Therefore, in data sets SA-YO and SA-LA, the evaluation values are dominated by those of the large data set, YOMIURI or LATIMES, respectively.

Table 1. Evaluation values for different sites data

Data Set	Optimal	Accuracy	Recall	Precision
WP-LA	$(28, 17)$	0.965 (0.969, 0.962)	0.802 (0.906, 0.679)	0.894 (0.859, 0.954)
SA-YO	$(19, 13)$	0.994 (0.892, 0.997)	0.889 (0.697, 0.930)	0.987 (0.965, 0.990)
SA-LA	$(23, 6)$	0.959 (0.914, 0.961)	0.958 (0.992, 0.954)	0.755 (0.796, 0.749)

Articles of the same site have the same formats. Data sets in Table. 1 have different formats since documents are collected from different sites. `FindOptimal` detects different formats as useless parts with high accuracy. Especially, it detects useless parts of articles in SANKEI despite of its small size.

4.4 Discussion

Many text mining algorithms assume that frequent patterns, substrings, rules, etc. are important. But, in this paper, frequent n-gram is defined to be useless. `FindOptimal` avoid treating an important keyword as a useless substring in the following way. `FindOptimal` increases the length n and frequency a from the initial cut point $(2, 1)$. Therefore, a frequent n-gram turns out to be long. In fact, $n = 24$ when WPOST is the input. We think that an important keyword is not so long and a substring for the structure and style is relatively long. Even if some important keywords are long, it merely happens that they appear frequently. Thus, the algorithm does not judge an important keyword to be useless.

This simple idea sometimes does not work well if the size of given input documents are not enough. For example, in YOMIURI data set, there exist some news files of Mr. Tsuta's death. He was a famous high-school baseball manager. In these news files, the substring "a former manager of the baseball club at Tokushima prefectural high-school" appears frequently. In Japanese, the length of the substing is 11, and the length of the cut point found by `FindOptimal` is 8. So, `FindOptimal` treats the substring to be useless although the substring is not for the structure or style of the documents. This type of errors does not happen when enough articles are given to `FindOptimal` because the frequency of such long substring become to be relatively low.

Note that useless parts do not consist of only tag sequences. For example, articles in YOMIURI have the same string in <title> tag and the string is detected as a useless part. On the other hand, the title of an article in WPOST is the same as the headline of the article and the title is detected as a non-useless part.

5 Conclusion

We introduced a new static value *alternation count* and developed an algorithm that divides each document of given documents into two parts, useless parts and non-useless parts. It is based on a simple assumption that frequent n-grams are not useful. We defined an optimal cut point to decide an appropriate pair (n, a) of length n and frequency a.

The algorithm does not depend on natural and markup languages because it counts substrings of inputs instead of words or other grammatical units. Experimental results show that the algorithm is robust for noise. Moreover, if input documents are collected from different sites and have different formats, an outputed cut point divides documents of both sites into useless and non-useless parts with hight accuracy.

We only showed experiments on news articles provided as HTML files. It is a future work to use other types of data sets such as dynamic pages, static pages except for news, files written in other markup language, etc.

It is an interesting future work to use some knowledge on grammars of the markup language. For example, when the algorithm enumerates n-grams, if a

delimiter such as "<", ">", ".", and "?", is contained in an n-gram, they must be at the beginning or end of the n-gram. This may improve accuracies.

As an application of the algorithm, we developed a record extraction system SCOOP [16]. A record extraction is an important application of Web mining [6, 9, 12]. SCOOP utilized `FindOptimal` as the preprocessor and knowledge that a delimiter of a record (or field) ends with ">" and begins with "<". Ginve an output of `FindOptimal`, SCOOP searches delimiters only near boundaries of non-useless parts and outputs the most frequent pair of substrings as a delimiter if it is unique on each record.

Another challenging future work is to apply our algorithm to genome infomatics. The *longest common subsequence problem* is, given two string, to find a longest common subsequence of them [2, 8]. The problem for k ($k \geq 2$) strings is known as *multiple sequence alignment*, which is a major problem in genome infomatics [11]. If k is not fixed, multiple sequence alignment is known to be NP-complete [13]. Both multiple sequence alignment and our problem are to extract parts common to a given set of strings. A difference between two problems is that each common part should have particular length in our setting, that is, our algorithm does not work well if common parts are too short.

References

1. S. Abiteboul, Querying Semi-structured Data. In *Proc. of the 6th International Conference on Database Theory*, pp. 1–18, 1997.
2. A. V. Aho, D. S. Hirschberg and J. D. Ullman, Bounds on the Complexity of the Longest Common Subsequences Problem. *J. ACM*, Vol. 23, No. 1, pp. 1–12, 1976.
3. R. Agrawal, T. Imielinski and A. Swami, Mining Association Rules between Sets of Items in Large Databases. In *Proc. of the 1993 ACM SIGMOD Conference on Management of Data*, pp. 207–216, 1993.
4. R. Agrawal and R. Srikant, Mining Sequential Patterns. In *Proc. of the 11th International Conference on Data Engineering*, pp. 3–14, 1995.
5. H. Arimura and S. Shimozono, Maximizing Agreement with a Classification by Bounded or Unbounded Number of Associated Words. In *Proc. of the 9th International Symposium on Algorithms and Computation*, Lecture Notes in Computer Science 1533, pp. 39–48, 1998.
6. P. Atzeni and G. Mecca, Cut and Paste. In *Proc. of the 16th ACM SIGMOD Symposium on Principles of Database Systems*, pp. 144–153, 1997.
7. CompletePlanet, The "Deep Web" White Paper. `http://www.completeplanet.com/Tutorials/DeepWeb/index.asp`.
8. M. Crochemore and W. Rytter, *Text Algorithms*. Oxford University Press, New York 1994.
9. D. W. Embley, Y. Jiang and Y. -K. Ng, Record-Boundary Discovery in Web Documents. In *Proc. of the 1999 ACM SIGMOD Conference*, pp. 467–478, 1999.
10. W. B. Frakes and R. Baeza-Yates (eds.), *Information Retrieval: Data Structures & Algorithms*. Prentice-Hall, 1992.
11. D. Gusfield, *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge University Press, New York, 1997.

12. N. Kushmerick, D. S. Weld and R. B. Doorenbos, Wrapper Induction for Information Extraction. In *Proc. of the 15th International Joint Conference on Artificial Intelligence*, pp. 729–737, 1997.
13. D. Maier, The Complexity of Some Problems on Subsequences and Supersequences. *J. ACM*, Vol. 25, No. 2, pp. 322–336, Apr. 1978.
14. H. Mannila and H. Toivonne, Discovering Generalized Episodes Using Minimal Occurrences. In *Porc. of the 2nd International Conference on Knowledge Discovery and Data Mining*, pp. 146–151, 1996.
15. M. F. Porter, An Algorithm for Suffix Stripping. *Automated Library and Information Systems*, Vol. 14, No. 3, pp. 130–137, 1980.
16. Y. Yamada, D. Ikeda and S. Hirokawa, SCOOP: A Record Extractor without Knowledge on Input. In *Proc. of the 4th International Conference on Discovery Science*, Lecture Notes in Artificial Intelligence, 2001. (to appear)