Gathering Text Files Generated from Templates

Ikeda, Daisuke
Kyushu University Library

Yamada, Yasuhiro
Department of Informatics, Kyushu University

Gathering Text Files Generated from Templates

Daisuke Ikeda

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

Yasuhiro Yamada

Department of Informatics, Kyushu University
Hakozaki 6-10-1, Fukuoka 812-8581, Japan.
y-yamada@i.kyushu-u.ac.jp
yshiro@matu.cc.kyushu-u.ac.jp

Abstract

Information integration comprises the three steps: data discovery; information extraction; and information integration. In this paper, we focus on the data discovery step which is crucial for the following steps. We first define what the data discovery is from the viewpoint of information extraction. The problem is, given a large amount of files, to find some sets of files such that found files in each set share some template. Each set corresponds to a template and multiple templates could be hidden in given files. We exploits a linear time algorithm which was originally developed by the authors for the common parts detection problem. The algorithm found different templates from collected Web pages including many noise files. We can cluster files according to the found templates. Files of a cluster is used as input data for an information extraction algorithm.

1 Introduction

The recent upsurge of data on the Web and its diversity pose the necessity of data integration. In the chaotic Web, however, many sites and persons provide series of data with the same type. For example, news articles on an online news outlet, blog entries, search result pages, catalogs on a shopping site, and so on. Once we can extract contents from such data and integrate them into a single database, we can easily find, compare, and utilize data on the Web. Imagine that we have an integrated database of cars of various makers and that you want to buy a sedan. All you need is just to search the database.

This scenario is completed by the following steps: data discovery; information extraction; and data integration. Information extraction has been well studied [2, 3, 5, 6, 8, 13, 15, 18]. Information integration has also been well studied, but the target is databases. Some researches on information integration for data on the Web appears recently [4, 10, 12, 16, 19]. In standard information extraction approaches, input files are assumed to have some similar structures and styles. However, it has not been considered how to find and collect them.

What are requirements for a data discovery algorithm? First, the algorithm has to find a site providing files with common structures. In [7], only clustering after this step is considered. In a found site, *all* files do not share the single structure in general. Moreover, several classes of structures could be hidden in a single site. So, the next task for the algorithm is to find templates among input files and to remove files without template. And finally, the algorithm clusters files according to the found templates. Thus the data discovery algorithm has to find, given files collected exhaustively by a crawler program, some sets of files such that files in each set share some template.

We can see that finding a common structure equals to finding some characteristic feature, such as machine learning, wrapper generation, data mining, clustering and so on. However, we can not exploit an algorithm of such a field for the data discovery since they are not so fast from the practical viewpoint and they require some background knowledge, such as the minimum support for data mining, training examples for machine learning, and the number of clusters for unsupervised clustering. But preparing appropriate parameters or examples is difficult when we consider data discovery since any data are not found yet.

As described above, it is necessary for a data discovery algorithm to collect files which share some features. But it is not necessary that such a feature completely describes collected data when we consider the data discovery for information extraction since to find more complex and descriptive features is a task after gathering data. Instead, it is necessary for a data discovery algorithm to be fast, scalable, and robust for noise since the algorithm must process a huge amount of data including noise files.

In this paper, we exploit an algorithm, called the *substring amplification*, for the data discovery. We adjust the algorithm since it was originally developed by the authors

**Proceedings of the 30th VLDB Conference,
Toronto, Canada, 2004**

to detect the common part among given text data [9, 11]. Although a found common part is simply defined as a set of common substrings, it is enough for a data discovery algorithm to express some common structure. The authors proved that the algorithm runs in linear time with respect to the total length of input files.

The algorithm calculates the total number $F(f)$ of occurrences of substrings appearing exactly f times in input files, and finds some f's providing extremely large $F(f)$ values. This amplifies the disparity of frequencies between substrings in a template and substrings in non-template parts. Therefore we see high peaks in an $F(f)$ graph (see Fig. 1) if input files contain common templates. Each peak corresponds to a template because a long substring in a template appears as many times as the number of files generated from the template. On the other hand, words or phrases in non-template parts hardly appear frequently since they are written in a natural language.

In [9, 11], given input files are assumed to be generated by a regular pattern [17] in the context of the machine learning of the pattern language [1]. A *pattern* is a string over constant and variable symbols. By substituting constant strings into all variables, a language is defined by a pattern. A pattern is *regular* if each variable appear at most once. The template for a pattern is a set of constant strings. For example, $p = axby$ is a regular pattern, where a and b are constants, and x and y are variables. Its template is $\{a, b\}$. The languages defined by p is $L = \{axby \mid x, y \in \{a, b\}^+\}$. We think a variable as a contents holder.

The authors considered only the case that input files are generated by a regular pattern. Many templates, however, may be hidden in input files and some type of contents appear repeatedly in one file. For example, a standard search result page contains 10 search results in one page. Thus we extend the substring amplification in Section 3.

We show two experimental results using data on the Web. They were collected by following links recursively from the top page of a site, and we have about 600 and 2,500 files, respectively. The extended substring amplification found many templates among input files including noise files. Some of the found templates are intended for whole-page layout so that files created by them look similarly. Other templates contain only local layout information, such as "HOME" and "BACK" buttons at the bottom of each page. The algorithm also found such a local template successfully. Using these template, we can cluster files and then files in a cluster can be an input for an information extraction algorithm.

2 Substring Amplification

In this section, we explain the substring amplification according to [9, 11].

The basic idea for the substring amplification is as follows. Consider that we have text files generated by some pattern. For example, news article files in which only contents parts are replaced. When there exist different

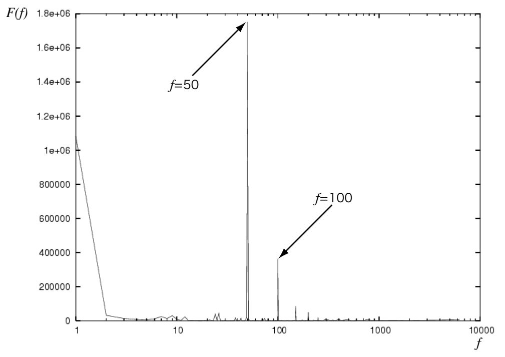


Figure 1: $F(f)$ graph for articles of "Sankei Shimbun"

types of contents, say three different types, we have $p = w_1 x w_2 y w_3 z w_4$, where each w_i is a constant string. Each type of contents is substituted into corresponding variable. Since contents are written in a natural language, there exist few possibilities that a long substring appears frequently if it is in contents parts. On the other hand, a long substring of w_i appears as many times as the number of the files. Thus we expect that we can find the template using frequencies of substrings if we have enough input files.

To count substrings, it is necessary to define the length to count in advance, otherwise just characters, which are the shortest substrings, appear frequently, and hide substrings we want to find. It is difficult, however, to decide an appropriate length before we get data files (see Section 5 for more discussion). Instead of deciding some fixed length, we count *all* substrings, sum up their frequencies, and calculate $F(f)$ for all f, where $F(f)$ is defined as follows. Let S be a set of strings[1] and $V(f)$ be a set of substrings which appear exactly f times in S. Then we define $F(f) = f \times |V(f)|$ so that $F(f)$ is the total number of occurrences of substrings appearing exactly f times in S.

Fig. 1 shows an $F(f)$ graph for 50 news files of "Sankei Shimbun[2]." These files are generated by some pattern p containing four different types of contents: date, headline, sub-headline, and body. 50 files were given to the algorithm without any modification (see Fig. 2).

We see that a clear peak at $f_p = 50$ which equals to the number of input files. A substring appearing exactly f_p times is a part of the hidden template. The peak is constituted by adding occurrences of all substrings appearing exactly f_p times. Therefore we call this method the substring *amplification*.

The template identified by $f_p = 50$ contains several substrings and each of them enough long. The string w in Fig. 2 is a part of the template between the date and the headline of an article. We see white spaces including newlines and multi-byte characters. 72 characters are included in w so that the number of all occurrences of substrings in w is $72 \cdot (72 - 1)/2 = 2556$. Moreover w is included in

[1]The substring amplification treats a file just as a string even if the file contains a structure like the tree structure of HTML tags.

[2]`http://www.sankei.co.jp/`

```
</b></i><br><P>
<center>
<table width=467>
<tr>
<td>

<!--------ヘッダ情報終了--------->

<!-----------★★ここから入れ替えてね・　・
---------------->
<font color="#8b0000">■</font><b>
```

Figure 2: String used for the template of “Sankei Shimbun”

all articles. Thus we have $F(50) = 2556 \times 50$. On the other hand, we have few substrings which are enough long and appear frequently in contents. For example, “The prime minister Koizumi” (26 characters) may appear frequently in news articles, but it contains only $26 \cdot (26 - 1)/2 = 325$ substrings. Due to the same reason, we can not expect that a single or a series of few tags become a whole template[3] although it appears frequently in HTML files.

We also see other peaks at $f = c \times 50$ ($c = 2, 3, \ldots$) since some substrings appear twice or more times in the template. For $f \neq c \times 50$ ($c = 1, 2, \ldots$), we have small $F(f)$ if f is enough large. This shows that a substring in a contents part is hard to appear frequently.

From these observations, we define the maximal peak.

Definition 1 *Let*

$$G(f) = \frac{F(f)}{(F(f-1) + F(f+1))/2},$$

where $F(f) = 1$ if $F(f)$ is undefined. Then we say that a frequency f_p provides the maximal peak if $G(f_p)$ is maximal among all $G(f)$ values.

From above observations, we can say that to find the maximal peak means to find a template.

To find the maximal peak, we have to count substrings. But it takes $O(n^2)$ time for a string with length n to count substrings directly since there exist $O(n^2)$ substrings. Instead, the substring amplification employs the suffix tree as the data structure.

The *suffix tree* for a string w is the compact trie for all suffixes of w [14]. For a node u of the tree, $BS(u)$ denotes the string obtained by concatenating all strings labeled on the edges of the path from the root to u. $BS(u)$ is called a *branching string*. The number of occurrences of $BS(u)$ in w equals to the number of leaves below u. For example, $BS(u) = ssi$ appears twice in *mississippi*\$[4] and u has two leaves in the suffix tree (see Fig. 3).

Let v be the parent of u. Obviously $BS(v)$ is a proper prefix of $BS(u)$. Moreover, let x be a prefix of $BS(u)$ which includes $BS(v)$ as a prefix. Then the numbers of occurrences of x and $BS(u)$ in w are the same. For example,

[3]It could be a *part* of a template, of course.

[4]The last character “\$” explicitly shows the end of the string.

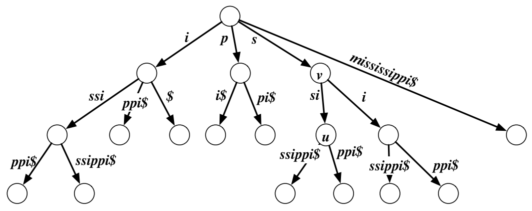


Figure 3: Suffix tree for *mississippi*\$

```
algorithm main(var S: set of strings):
    set of strings
  var
    V, F: hash table;
  begin
    V:=Count(S);
    for f in keys(V);
       F(f):=0;
       for w in V(f);
         F(f) += f|w|;
       end;
    end ;
    f:=FindPeaks(F);
    return(V(f));
  end
```

Figure 4: A pseudo code of the substring amplification

both $BS(u) = ssi$ and ss appear twice. Therefore when we count substring frequencies, all we have to do is to count only branching strings.

In a suffix tree, a node corresponds to a frequency. This fact leads the followin key lemma since the number of nodes in a suffix tree is $O(n)$.

Lemma 1 *There exist at most $O(n)$ different frequencies of substrings, where n is the total length of S.*

Fig. 4 is a pseudo code of the substring amplification. An input is a set S of strings and an output is a set of branching strings in S. A template which generates S consists of these branching strings.

`Count(S)` counts frequencies of $BS(u)$ for each node u in the suffix tree for all strings in S. This is done in $O(n)$ time. And then the subroutine creates a hash table V in which a key is a frequency f and a value is $V(f)$.

The algorithm then calculates $F(f)$ from $V(f)$. We need to count frequencies for both non-branching and branching strings although V contains those for only branching strings. It takes also $O(n)$ time to compute $F(f)$ since calculation `F(f)+=f|w|` is totally executed as many times as the number of branching strings.

Then the algorithm calls `FindPeaks(F)` which returns a frequency providing the maximal peak. From Lemma 1, there exist at most $O(n)$ different frequencies. Therefore this routines done in $O(n)$ time.

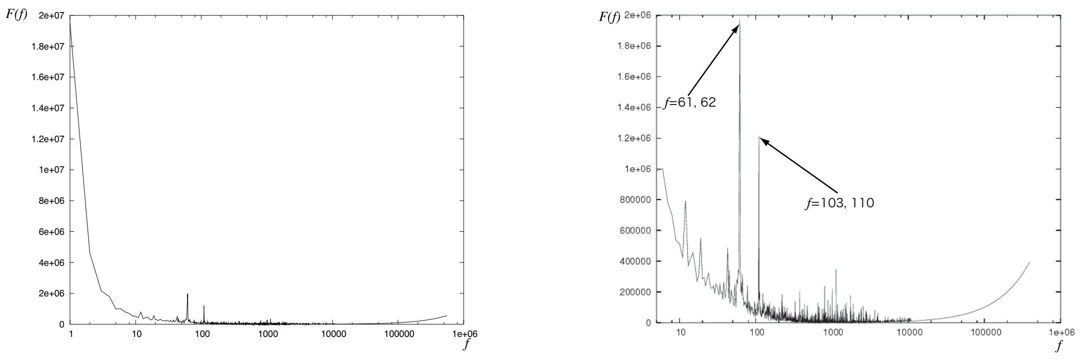


Figure 5: $F(f)$ graphs for Web pages in Kyushu University, where $f \geq 1$ (left) and $f \geq 5$ (right)

Theorem 1 ([9, 11]) *The time complexity of the algorithm in Fig. 4 is $O(n)$, where n is the total length of strings in S.*

3 Data Discovery with the Substring Amplification

When we consider a single template, all we have to do is to find the highest peak only. But in the case of the data discovery, we have to find multiple sets of files generated from different templates. This means that several peaks must be considered. We introduce a threshold value δ and redefine a peak as follows.

Definition 2 *Given a threshold δ, we say that a frequency f provides a peak if $G(f) > \delta$.*

By the above definition, a user must give a threshold value. But this is not a hard burden for the user since the substring amplification does not employ any pruning techniques and so the user can change the value dynamically even after counting substrings without any additional costs.

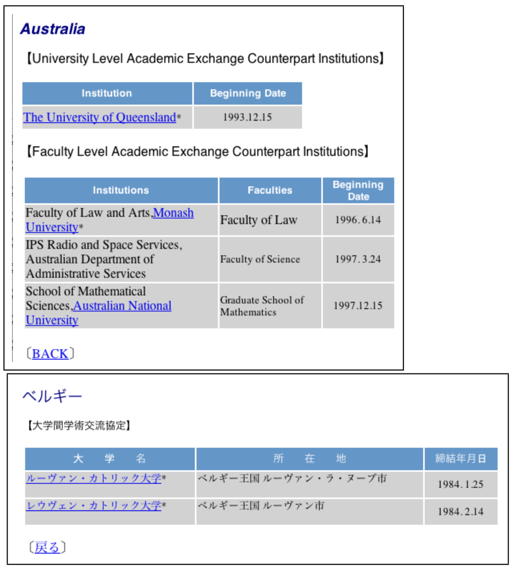
Next we extend the algorithm. We give a positive number as δ to the algorithm in addition to S. We also give δ to the `FindPeaks(F, δ)` which now returns multiple frequencies providing peaks. Since `FindPeaks(F)` calculates $G(f)$ for each f, obviously the time complexity is not changed.

Corollary 1 *The time complexity of the extended algorithm is also $O(n)$.*

In the previous section, it is mentioned that it is rare for a substring to appear frequently if it is in contents parts. This also holds even if input files generated by multiple templates. Moreover, a long substring in a template of the templates appears as many times as the number of files generated by the template. Therefore substrings in the template appear frequently. Thus we can again find multiple templates by frequencies of substrings.

4 Experiments

In this section, we show two experimental results using text data on the Web.



Australia

【University Level Academic Exchange Counterpart Institutions】

Institution	Beginning Date
The University of Queensland*	1993.12.15

【Faculty Level Academic Exchange Counterpart Institutions】

Institutions	Faculties	Beginning Date
Faculty of Law and Arts,Monash University*	Faculty of Law	1996.6.14
IPS Radio and Space Services, Australian Department of Administrative Services	Faculty of Science	1997.3.24
School of Mathematical Sciences,Australian National University	Graduate School of Mathematics	1997.12.15

〔BACK〕

ベルギー

【大学間学術交流協定】

大　学　名	所　在　地	締結年月日
ルーヴァン・カトリック大学*	ベルギー王国 ルーヴァン・ラ・ヌーブ市	1984.1.25
レウヴェン・カトリック大学*	ベルギー王国 ルーヴァン市	1984.2.14

〔戻る〕

Figure 6: HTML files generated from the template identified by $f = 57$ (English and Japanese)

First, we give 598 HTML files (5584 Kbytes) into the algorithm. They were collected by following links recursively at most three depth from the top page of Kyushu university[5]. Kyushu university has many schools, institutes, faculties, and departments. They have their own Web pages independently. In this sense, we can say that gathering pages at a large university, like Kyushu university, means gathering pages among different sites.

Fig. 5 is $F(f)$ graphs for these files. Compared to the graph in Fig. 1, peaks are smaller and irregular in the left graph in Fig. 5.

The template identified by $f = 57$ is used for Web pages of international academic exchanges (see Fig. 6). These pages are written in English or Japanese. Each page contains table(s), but the number of tables are different between pages. The substring amplification is not affected by neither the contents language nor the number of instances.

The template identified by $f = 62$ is used in 62 pages. One of them is the top page of Kyushu university (the left most picture in Fig. 7) and the same style is also used for

[5]`http://www.kyushu-u.ac.jp/`

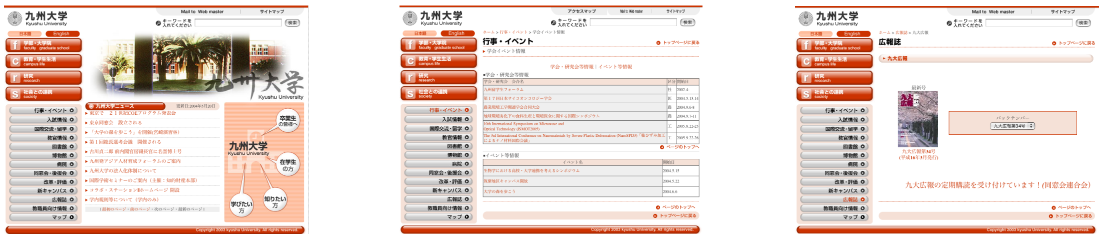


Figure 7: HTML files generated from the template identified by $f = 62$. Navigation links at left-hand side and the search form at the upside are the common among these files

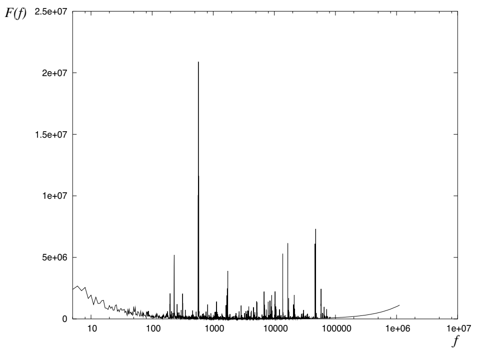


Figure 8: $F(f)$ graphs for Web pages in Sankei Shimbun, where $f \geq 5$

the other two pictures in Fig. 7.

The template of $f = 28$ consists of only the background color, "HOME" button, and "BACK" button. The buttons are at the bottom of each page. The substring amplification can find such small and local characteristics among a lot of Web pages.

Second, we give 2495 HTML files (53.8 Mbytes) collected from the top page of "Sankei Shimbun." Compared to the previous experiment, the data size is quite large. Fig. 8 is the $F(f)$ graph for these files. The substring amplification also found many templates among these files. The maximal peak is at $f = 572$.

The above results show necessity for data discovery even when we consider files from a single site since different templates are used even in a single site and not all files gathered from a single site share the templates.

5 Discussion

In our setting, we count substrings with any length. On the other hand, in n-gram statistics, which is a major framework in natural language processing, we count substrings with length n or series of n words, where n is fixed. When we count n words, we need to do morphological analysis although we can not know what kind of language describes contents. The former case (n characters) involves another difficult problem. For a small n, substrings with high frequencies are often *stop words*. Therefore we need a list of stop words and remove them. But the list depends on languages. Thus this framework requires to have some background knowledge. Instead, the substring amplification counts *all* substrings and calculates $F(f)$. A longer substring gives a large $F(f)$ which negates $F(f)$ for short stop words.

Frequent pattern mining also involves a similar problem. To avoid this, in this framework, a user must provide a threshold value to a pattern mining algorithm. Unlike the substring amplification, a user can not change this value dynamically since a frequent pattern mining algorithm prunes its search space using the threshold.

In general, files sharing with some template in a site are stored in one directory. Therefore you expect that a data discovery algorithm can collect such files by only the string processing of URLs, such as an algorithm in [7]. Such a heuristics method, however, does not guarantee that collected files are *all* files which share the template. In fact, the proposed algorithm found Web pages which are stored in different directories. On the other hand, the substring amplification is guaranteed to collect *all* files sharing the template exhaustively even if input files are collected from different sites.

6 Conclusion

We showed that the extended substring amplification can find groups of files such that files in each group are generated by some common template. Therefore our algorithm is applicable to the first step of information integration, data discovery.

The substring amplification is an unsupervised clustering algorithm. The algorithm uses a frequency f as its feature. This enables the substring amplification to be a linear time algorithm. Moreover, the algorithm finds multiple templates even if the number of templates are not fixed in advance and input files are very noisy.

Using this algorithm, we can gather files written in any formats and any languages since the algorithm treats a file just a string, and counts all substrings instead of some grammatical units, such as tags of HTML/XML or words in some natural language. In the data discovery step, we can not know what kinds of formats or languages are used. Thus independence from formats and languages is important for the data discovery step.

We assumed that a long substring does not appear in any contents part frequently. This means that such a long substring may appear frequently with some possibility. But this is not a problem since the probability is enough small from the practical viewpoint [9, 11].

We only showed experiments using files gathered from only one site. But the substring amplification does not have any limitation on the number of sites, and is applicable to discovery among different sites. It is an interesting future work to find blog sites or sites with search facilities, which contain useful contents and they are created by some fixed softwares, such as Movable Type[6] and Namazu[7].

References

[1] Dana Angluin. Finding Patterns Common to a Set of Strings. *Journal of Computer and System Sciences*, 21:46–62, 1980.

[2] Arvind Arasu and Hector Garcia-Molina. Extracting Structured Data from Web Pages. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, pages 337–348, 2003.

[3] Chia-Hui Chang and Shao-Chen Lui. IEPAD: Information Extraction Based on Pattern Discovery. In *Proceedings of the 10th International World Wide Web Conference*, pages 4–15, 2001.

[4] Boris Chidlovskii. Schema Extraction from XML Collections. In *Proceedings of the second ACM/IEEE Joint Conference on Digital Libraries*, pages 291–292, 2002.

[5] William W. Cohen and Lee S. Jensen. A Structured Wrapper Induction System for Extracting Information from Semi-structured Documents. In *Proceedings of IJCAI 2001 Workshop on Adaptive Text Extraction and Mining*, 2001. `http://www.smi.ucd.ie/ATEM2001/proceedings/`.

[6] Valter Crescenzi, Giansalvatore Mecca, and Paolo Merialdo. RoadRunner: Towards Automatic Data Extraction from Large Web Sites. In *Proceedings of the 27th International Conference on Very Large Data Bases*, pages 109–118, September 2001.

[7] Valter Crescenzi, Giansalvatore Mecca, and Paolo Merialdo. Wrapping-Oriented Classification of Web Pages. In *Proceedings of the 2002 ACM symposium on Applied Computing*, pages 1108–1112, 2002.

[8] David W. Embley, Y. S. Jiang, and Yiu-Kai Ng. Record-Boundary Discovery in Web Documents. In *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*, pages 467–478, 1999.

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

[10] Daisuke Ikeda. Instance Based Table Integration Algorithm for Multilingual Tables on the Web. *Bulletin of Informatics and Cybernetics*, 35(1–2), 2004. (to appear).

[11] Daisuke Ikeda, Yasuhiro Yamada, and Sachio Hirokawa. A Pattern Discovery Algorithm by Substring Amplification. *IPSJ Transactions on Mathematical Modeling and Its Applications*, 2004. (in Japanese, to appear).

[12] Craig A. Knoblock, Steve Minton, Jose Luis Ambit, Naveen Ashish, Pragnesh Jay Modi, Ion Muslea, Andrew G. Philpot, and Sheila Tejada. Modeling Web Sources for Information Integration. In *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI-98)*, pages 211–218, 1998.

[13] Nicholas Kushmerick, Daniel S. Weld, and Robert B. Doorenbos. Wrapper Induction for Information Extraction. In *Proceedings of the 15th Intenational Joint Conference on Artificial Intelligence*, pages 729–737, 1997.

[14] Edward M. McCreight. A Space-Economical Suffix Tree Construction Algorithm. *JACM*, 23(2):262–272, 1976.

[15] Ion Muslea, Steve Minton, and Craig A. Knoblock. STALKER: Learning Extraction Rules for Semistructured Web-based Infomation Sources. In *Proceedings of AAAI-98 Workshop on AI and Information Integration*, pages 74–81, 1998.

[16] Lucian Popa, Yannis Velegrakis, Renee J. Miller, Mauricio A. Hernández, and Ronald Fagin. Translating Web Data. In *Proceedings of the 28th International Conference on Very Large Data Bases*, pages 598–609, August 2002.

[17] Takeshi Shinohara. Polynomial Time Inference of Extended Regular Pattern Languages. In *RIMS Symposium on Software Science and Engineering (1982)*, Lecture Notes in Computer Science 147, pages 115–127. Springer-Verlag, 1983.

[18] Yasuhiro Yamada, Daisuke Ikeda, and Sachio Hirokawa. Automatic Wrapper Generation for Multilingual Web Resources. In *Proceedings of the 5th International Conference on Discovery Science*, Lecture Notes in Computer Science 2534, pages 332–339. Springer-Verlag, November 2002.

[19] Minoru Yoshida, Kentaro Torisawa, and Jun'ichi Tsujii. A Method to Integrate Tables of the World Wide Web. In *Proceedings of the 1st International Workshop on Web Document Analysis (WDA 2001)*, pages 31–34, 2001.

[6] `http://www.movabletype.org/`

[7] `http://www.namazu.org/index.html.en`