Automatic Wrapper Generation for Multilingual Web Resources

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

Ikeda, Daisuke
Computing and Communications Center, Kyushu University

Hirokawa, Sachio
Computing and Communications Center, Kyushu University

出版情報：Lecture Notes in Computer Science. 2534, pp.107-113, 2002-11. Springer
バージョン：

Automatic Wrapper Generation for Multilingual Web Resources

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

[1] Graduate School of Information Science and Electrical Engineering,
Kyushu University, Fukuoka 812-8581, Japan
yshiro@matu.cc.kyushu-u.ac.jp

[2] Computing and Communications Center,
Kyushu University, Fukuoka 812-8581, Japan
{daisuke, hirokawa}@cc.kyushu-u.ac.jp

Abstract. We present a wrapper generation system to extract contents of semi-structured documents which contain instances of a record. The generation is done automatically using general assumptions on the structure of instances. It outputs a set of pairs of left and right delimiters surrounding instances of a field. In addition to input documents, our system also receives a set of symbols with which a delimiter must begin or end. Our system treats semi-structured documents just as strings so that it does not depend on markup and natural languages. It does not require any training examples which show where instances are. We show experimental results on both static and dynamic pages which are gathered from 13 Web sites, markuped in HTML or XML, and written in four natural languages. In addition to usual contents, generated wrappers extract useful information hidden in comments or tags which are ignored by other wrapper generation algorithms. Some generated delimiters contain whitespaces or multibyte characters.

1 Introduction

There are useful information hidden in enormous pages on the Web. It is difficult, however, to extract and restructure them because these pages do not have an explicit structure like database systems. To use pages on the Web like a database system, it is necessary to extract contents of pages as records or fields.

A wrapper is a procedure to extract instances of records and fields from Web pages. A database consists of some records, and a record consists of some fields. An instance is an instantiated object of a record or field. For example, in result pages of a typical search engine, a record is a tuple (page title, caption, URL), a field is an element of a record.

Thinking of the enormous pages on the Web, it is hard to generate wrappers manually. Basically, there are three approaches to generate wrappers. The first approach is based on machine learning [6, 7] using training examples. A problem of machine leaning approaches is that making training examples is too costly.

The second approach is to assume input documents are only HTML and use knowledge on HTML [2, 4]. In [4], record boundaries are determined by combination of heuristics one of which is a boundary is near some specific tags. This approach does not require any training examples, but this is not applicable to other markup languages.

The third approach exploits regularity of input documents instead of background knowledge or training examples. IEPAD [3] tries to find record separators using the maximum repeat of a string. The data extraction algorithm in [8] also finds regularity of lists in input HTML files. Our system, similarly, determines common parts in given documents, then finds delimiters on common parts. A superiority of our system is to find common parts *roughly* and to be applicable to data with some irregularity.

The authors developed a prototype of contents extraction system, called SCOOP [9]. It is based on a very simple idea that frequent substrings of input documents are useless and are not contents. Like other wrapper generation systems, SCOOP also has problems in Section 1.1. The main contribution of this paper is to propose, based on SCOOP, a full automatic wrapper generation system without any training examples. An input for the system is a set of symbols, called *enclosing symbols*, and a set of semi-structured documents containing instances of a record. A generated wrapper is an LR wrapper [6, 7].

We show experimental results in Section 3. Input files are HTML and XML files gathered from 13 sites, and contents of them are written in four languages (Chinese, English, German, and Japanese). A generated wrapper extracts instances of fields with high accuracy. It also extracts useful information hidden in comments or tags which are ignored by other wrapper generation algorithms.

1.1 Our Contributions

Multilingual System: Although Web resources are written in many languages, many other wrapper generation systems are mono- or bilingual. Our system treats input semi-structured documents just as strings, so that it is multilingual[3] in two meanings, for markup and natural languages. In the near future, XML files will become widespread on the Web. But a wrapper from XML files has not been considered because they have explict structures by nature. Since restructuring of semi-structured documents is an important goal of wrapper generation, it is important to generate wrappers from XML files.
Dynamic and Static Pages: The target of other wrapper generation algorithms is a set of dynamic pages. Dynamic pages are created automatically by some database programs or search facilities. Dynamic pages ideally have completely the same template, so that such pages seem to be easy to generate wrappers. But, in practice, dynamic pages of a site have some irregularities. This is one of most difficult problem of wrapper generation systems.

Since static pages usually have larger irregularities than dynamic ones, a wrapper generation system which works well for static pages also can be expected to work well for dynamic pages with some irregularities. Therefore, wrappers are important for both static and dynamic pages. SCOOP [9] can make a wrapper

[3] So is SCOOP [9], but its implementation is bilingual (English and Japanese).

from such static pages, but it can not handle dynamic pages. The presented system is good at both static and dynamic pages.

The Number of Instances: In an address book, for example, some does not have an email address, and other have some email addresses. More generally, we must consider the case that some instances of a record have different number of instances of a field. In SCOOP[9], instances in a field must be instantiated from different fields. In other words, all people in the address book must have at most one email address. The presented system overcomes this problem.

2 Main Algorithm

Our wrapper generation algorithm receives a set of semi-structured documents including some instances of a record. It treats each semi-structured document as just string. It also receives E_l and E_r, where E_l and E_r are sets of symbols called *enclosing symbols*. It outputs a set of rules extracting instances of each field.

The algorithm consists of three stages, contents detection, rule extraction, and deleting and integrating rules. In contents detection stage, it divides roughly each input string into common and uncommon parts.

In rule extraction stage, it extracts a set of rules. Roughly speaking, a rule is a pair of delimiters, called a *left delimiter* and a *right delimiter*. A left delimiter is a string ending with a symbol in E_l and a right delimiter is a string beginning with a symbol in E_r. We define the *length* of a delimiter to be the number of enclosing symbols. A *rule* is a pair (l, r) of left and right delimiters such that l and r have the same number of occurrences on each input string.

In deleting and integrating rules stage, it deletes useless rules. It is difficult to decide whether a field is useful or not. So we assume that a field is useless if only less than half of input documents have instances of it. Finally, it integrates rules extracting the same string and treats them as a rule.

2.1 Contents Detection

In this stage, our wrapper generation algorithm divides each input string into two parts roughly, common and uncommon parts. It utilizes the algorithm `FindOptimal` developed in [5]. Our algorithm makes full use of the fact that uncommon parts of semi-structured documents well cover contents [5].

In [5], it is experimentally shown that, given news articles written in English or Japanese gathered from a news site, `FindOptimal` extracts contents with high accuracy – more than 97%.

The original `FindOptimal` preprocesses given strings. It converts successive whitespaces into a space because whitespaces are ignored when HTML files are displayed by a browser. The current version uses given strings as they are.

2.2 Rule Extraction

In this stage, the algorithm receives a set of strings, a set of common and uncommon divisions of strings, and a set of enclosing symbols.

For each uncommon part, the algorithm finds two enclosing symbols l_e and r_b such that they cover whole the uncommon part and they are the nearest from the uncommon part. The first candidate of a left delimiter ends with l_e and begins with the previous enclosing symbol. Similarly, the first candidate of a right delimiter begins with r_b and ends with the next enclosing symbol.

If two candidates have different numbers of occurrences, then the algorithm increases the length of the frequent candidate. If l_e (r_b) is more frequent than r_b (l_e), then it increases the length of the left (right) candidate until the previous (next) enclosing symbol. It continues this until the occurrence of left and right candidates (l, r) are the same.

If l and r are the same string or they are corresponding tags (e.g., $l =$ <tagA><tagB> and $r =$ </tagB></tagA>), the algorithm increases the length of both candidates and checks the number of their occurrences.

2.3 Deleting and Integrating Rule

Let R be a set of candidates for rules. It is necessary to delete and integrate candidates in R because some of them extract the same string or other of them are useless.

In our setting, a rule is allowed to extract no instances of a field from some input strings. We put a restriction on a rule such that it must extract instances from more than half of input strings. Otherwise the algorithm deletes it from R.

Next, it integrates candidates on R extracting the same string from each string. For example, if these two candidates, (<P><P>, </P>\n) and (␣␣␣ <P><P> ␣, </P>\n␣), extract the same string from each input string, it integrates these two candidates and treats them as a rule.

3 Experiments

We implement the algorithm described in the previous section in Python. Input files are HTML and XML files, and contents of them are written in four languages (Chinese, English, German, and Japanese). They are gathered from 13 sites (see Table 1) and the number of all gathered files is 1197. We set E_l and E_r are sets of ">" and a whitespace (space, tab, newline characters), and "<" and a whitespace, respectively.

To evaluate the results, for each site the authors see some HTML/XML sources in advance and create a wrapper manually. Then we compare two extraction results from wrappers created manually and automatically.

Table 2 and Table 4 have results of static and dynamic pages. The second column "Field (Accuracy)" has attribute names of fields in hand-coded wrappers, that is, the fields expected to be extracted and their accuracies. Ev_1 shows the number of fields which authors overlook when they created wrappers manually. Ev_2 shows the number of fields extracted wrongly, so we want Ev_2 to be small.

Table 1. URL list of 13 sites described in this section. The fourth column stands for the number of files. We gathered 1197 files from the sites. "Search" and "Mail" in Type column mean that these pages are result pages of search engines and mail archives, respectively. "News" pages are gathered from online news sites. "Manual" stands for online manual pages. "Database" means that we got data from some public database. "kyushu-u" is now under-construction and not public yet

ID	URL	Language	#	Type
	HTML			
altavista	http://www.altavista.com/	English	17	Search
freebsd	http://docs.freebsd.org/mail/	English	49	Mail
ftd	http://www.ftd.de/	German	101	News
java	http://java.sun.com/j2se/1.3/docs/	English	30	Manual
lycos	http://www.lycos.com/	English	50	Search
peopledaily	http://www.peopledaily.co.jp/	Chinese	127	News
redhat	http://www.redhat.com/mailing-lists/	English	50	Mail
reuters	http://www.reuters.de/	German	50	News
sankei	http://www.sankei.co.jp/main.htm	Japanese	108	News
yahoo	http://www.yahoo.com/	English	45	Search
	XML			
kyushu-u	–	Japanese	50	Database
mainichi	http://www.mainichi.co.jp/digital/newsml/	Japanese	470	News
sigmod	http://www.acm.org/sigmod/record/xml/	English	50	Database

3.1 Static Pages

As described in Section 1.1, most of other wrapper generation algorithms assume that input documents are created dynamically. Such dynamic pages are created by filling a template so that common parts created by one template are completely same. So, it is difficult to create wrappers from static pages than from such dynamic ones. Table 2 shows results of static pages and our algorithm works well for such pages.

Table 3 shows the wrapper created on "mainichi" which is a set of XML files. We can see that tags in the table are completely different from those of HTML. Our algorithm finds rules for two "date" and two "keyword" fields. We can see in rules whitespaces which are just for readability of XML sources. In [5, 9], successive whitespaces are compressed into a space, so SCOOP in [9] can not find such a rule.

The algorithm fails to find a rule for "Body text" field. A body text in "mainichi" is in between "\n<p>" and "\n<p>". They are the same, so the system tries to find delimiters with longer lengths. However, the right delimiter is followed by date which is variable, so that the system fails to find a good right delimiter. Other 0% fields in Table 2 also occur by similar problems.

Our system succeeds to find the field "second headline" from "sankei" although there are variations of the number of its instances: 26 files have no in-

Table 2. Results of static pages

ID	Field (Accuracy)	Ev_1	Ev_2
ftd	page title (0%), headline (100%), summary of article (100%), body (100%)	6	5
java	classname (90%), date (100%), return (100%), body (100%)	1	2
mainichi	headline (100%), date (100%), keyword (100%), body (0%), related word (100%), other headline (100%)	1	0
peopledaily	page title (100%), date (100%), headline (98.4%), body (99.2%)	4	2
reuters	headline (100%), date (100%), body (100%)	4	3
sankei	headline (100%), second headline (100%), body (100%)	0	0

stances of “second headline”[4], 21 files have two instances, 2 files have three instances, and the other files have one instance. Our system does not mind the

Table 3. A part of wrapper created by our system from “mainichi”

Field	Wrapper
Date1	</HeadLine>\n\t\t\t\t<DateLine>
	</DateLine>\n\t\t\t\t</CreditLine␣xml:lang=”ja”>
Date2	␣(␣
	␣)</p>
Keyword	\n\t\t\t\t<KeywordLine>
	</KeywordLine>\n\t\t\t\t</NewsLines>
Keyword2	<midasi>\n\t\t\t\t\t\t</kanrenmidasi>
	</kanrenmidashi>\n\t\t\t\t\t\t</midasibun>
Body text	can not extract rule

number of instances of a field (see Section 1.1). From this data set, the system finds a rule whose left delimiter contains a multibyte character, “<font color=”#8b0000”>■</font><b>”, where “■” is a multibyte character and used for the header symbol for a headline.

Some useful contents hidden in meta tags or comments are extracted. An article in “ftd” contains a brief summary of the article in a meta tag, and the date of the article in a comment tag.

In Table 2, there are some fields whose accuracies are high but not perfect. The reason of partial failure is in input files. We assume instances of a field are surrounded by the same pair of strings. But, sometimes there are some files which have instances surrounding by other strings. For example, most instances of a field are followed by “<abc” but the other instances by “<ABC”. Our

[4] In these files, there exist no delimiters for the second headline.

algorithm fails to extract contents from the latter instances because it searches strings case-sensitively.

3.2 Dynamic Pages

A typical dynamic page is a search result. We select three major search engines and two mail archives (see Table 4). A search result contains the title of a found page, URL, and brief description of the page. A typical page of mail archives contains the body of a found mail, subject, and some mail headers. Table 4

Table 4. Results of dynamic pages

ID	Field (Accuracy)	Ev_1	Ev_2
altavista	title of page (100%), caption (100%), URL (100%)	7	4
lycos	title of page (100%), caption (95.5%), URL (100%)	5	3
yahoo	title of page (100%), caption (100%), URL (100%)	7	11
freebsd	Date (100%), From (100%), To (93.9%), Subject (100%), Message-ID (100%), content (100%)	0	2
redhat	Subject (100%), content (98%)	3	8

shows the presented algorithm treats such pages well although SCOOP [9] failed to find rules.

We also have results of the following two databases: “sigmod” and “kyushu-u.” They consist of XML files. “sigmod” is gathered from “OrdinaryIssuePage” in “ACM SIGMOD Record: XML Version.” A record has following fields: title of the article, author name(s), volume, number, year, start and end pages. All of these fields are completely found by our algorithm. It also successfully finds unique ID number in an XML tag and created time in a comment.

“kyushu-u” stands for a database of academics in Kyushu university. A file corresponds to an academic’s record. A record contains his/her name, affiliation, major, mail address, publications, classes and so on. An XML file of this data set has tags including Japanese characters, but it is not a problem for our system. It found rules containing tags of Japanese characters.

4 Conclusion

We presented a simple wrapper generation algorithm. Any additional inputs are not necessary except that enclosing symbols each of which is the first or last letter of a delimiter. The system is suitable for any semi-structured documents. This is due to simplicity of our system: it treats input documents just as strings and utilizes regularity of instances.

Extraction is successful for both dynamic and static pages while SCOOP in [9] failed to find rules from dynamic pages because SCOOP depends heavily on `FindOptimal` in [5] and `FindOptimal` fails to divide dynamic pages well into common and uncommon parts. Our system also found useful information hidden in comments and attribute values of tags, such as meta tags. Presented experiments show that whitespaces play an important role for structuring data. These results contrast to other wrapper generation algorithms because they ignore inside of tags and comments, and whitespaces.

However, sometimes the system failed to find delimiters. Typically, this happens when instances of a field are surrounded by simple corresponding tags, such as, "<B>" and "</B>". Such corresponding tags frequently appear in HTML documents and has nothing to do with the structure we want to extract. Therefore, we do not treat such tags as a rule. It will be a good solution to use tree wrappers instead of string based wrappers.

An important future work is to combine extracted fields into one record. In [8], the same problem was discussed. However, our case is more difficult because the setting in [8] is that each file must have multiple instances of a record. Our wrapper generation system can deal with both single and multiple cases.

References

1. N. Ashish and C. Knoblock, Wrapper Generation for Semi-structured Internet Sources, Proc. of *Workshop on Management of Semistructured Data*, 1997.
2. D. Buttler, L. Liu and C. Pu, A Fully Automated Object Extraction System for the World Wide Web, International Conference on Distributed Computing Systems, 2001.
3. C.-H. Chang and S.-C. Lui, IEPAD: Information Extraction Based on Pattern Discovery, Proc. of *the Tenth International Conference of World Wide Web (WWW2001)*, pp. 4–15, 2001.
4. D. W. Embley, Y. Jiang and Y. -K. Ng, Record-Boundary Discovery in Web Documents, Proc. of *ACM SIGMOD Conference*, pp. 467–478, 1999.
5. D. Ikeda, Y. Yamada and S. Hirokawa, Eliminating Useless Parts in Semi-structured Documents using Alternation Counts, Proc. of *the Fourth International Conference on Discovery Science*, Lecture Notes in Artificial Intelligence, Vol. 2226, pp. 113–127, 2001.
6. N. Kushmerick, D. S. Weld and R. B. Doorenbos, Wrapper Induction for Information Extraction, Intl. Joint Conference on Artificial Intelligence, pp. 729–737, 1997.
7. N. Kushmerick, Wrapper Induction: Efficiency and Expressiveness, Artificial Intelligence, Vol. 118, pp. 15–68, 2000.
8. K. Lerman, C. A. Knoblock and S Minton, Automatic Data Extraction from Lists and Tables in Web Sources, Adaptive Text Extraction and Mining workshop, 2001.
9. Y. Yamada, D. Ikeda and S. Hirokawa, SCOOP: A Record Extractor without Knowledge on Input, Proc. of *the Fourth International Conference on Discovery Science*, Lecture Notes in Artificial Intelligence, Vol. 2226, pp. 428–487, 2001.