Efficient Record-Level Wrapper Induction

Shuyi Zheng* [1]   Ruihua Song[2]   Ji-Rong Wen[2]   C. Lee Giles[1,3]

[1]Department of Computer Science and Engineering
Pennsylvania State University, University Park, PA 16802, USA

[2]Microsoft Research Asia, Beijing 100190, China

[3]College of Information Sciences and Technology
Pennsylvania State University, University Park, PA 16802, USA

shzheng@cse.psu.edu   {rsong,jrwen}@microsoft.com   giles@ist.psu.edu

ABSTRACT

Web information is often presented in the form of record, e.g., a product record on a shopping website or a personal profile on a social utility website. Given a host webpage and related information needs, how to identify relevant records as well as their internal semantic structures is critical to many online information systems. Wrapper induction is one of the most effective methods for such tasks. However, most traditional wrapper techniques have issues dealing with web records since they are designed to extract information from a page, not a record. We propose a record-level wrapper system. In our system, we use a novel "broom" structure to represent both records and generated wrappers. With such representation, our system is able to effectively extract records and identify their internal semantics at the same time. We test our system on 16 real-life websites from four different domains. Experimental results demonstrate 99% extraction accuracy in terms of F1-Value.

Categories and Subject Descriptors

H.3.m [**Information Storage and Retrieval**]: Miscellaneous—*Data Extraction, Wrapper Generation, Web*

General Terms

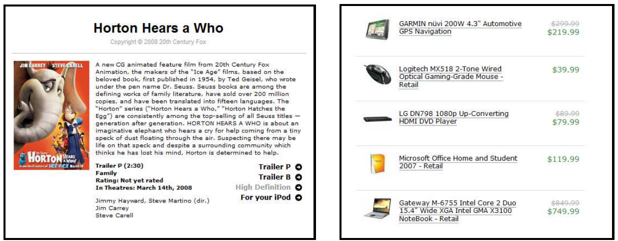
Algorithms, Experimentation, Performance

Keywords

Wrapper, Information Extraction, Web Record, Broom Representation

1. INTRODUCTION

The amount of Web information has been increasing rapidly, especially with the emergence of Web 2.0 environments, where users are encouraged to contribute rich content. Much Web information is presented in the form of a *Web record* which exists in both detail and list pages. For example, in Figure 1(a), a movie record is presented in a detail page from `www.apple.com`; in Figure 1(b), five product records are presented in a list page from `www.newegg.com`.



Horton Hears a Who

Copyright © 2008 20th Century Fox

A new CG animated feature film from 20th Century Fox Animation, the makers of the "Ice Age" films, based on the beloved book, first published in 1954, by Ted Geisel, who wrote under the pen name Dr. Seuss. Seuss books are among the defining works of family literature, have sold over 200 million copies, and have been translated into fifteen languages. The "Horton" series ("Horton Hears a Who," "Horton Hatches the Egg") are consistently among the top-selling of all Seuss titles — generation after generation. HORTON HEARS A WHO is about an imaginative elephant who hears a cry for help coming from a tiny speck of dust floating through the air. Suspecting there may be life on that speck and despite a surrounding community which thinks he has lost his mind, Horton is determined to help.

Trailer P (2:30)
Family
Rating: Not yet rated
In Theatres: March 14th, 2008

Jimmy Hayward, Steve Martino (dir.)
Jim Carrey
Steve Carell

(a) Detail Page

GARMIN nüvi 200W 4.3" Automotive GPS Navigation	~~$299.99~~ $219.99
Logitech MX518 2-Tone Wired Optical Gaming-Grade Mouse - Retail	$39.99
LG DN798 1080p Up-Converting HDMI DVD Player	~~$89.99~~ $79.99
Microsoft Office Home and Student 2007 - Retail	$119.99
Gateway M-6755 Intel Core 2 Duo 15.4" Wide XGA Intel GMA X3100 NoteBook - Retail	~~$849.99~~ $749.99

(b) List page

Figure 1: Example of Web Records in a Detail Page and a List Page

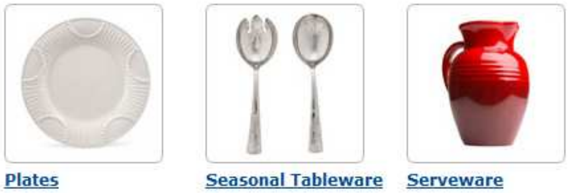
Although Web records are normally rendered from a structured source with semantic definitions, e.g., a back-end relational database, they are often presented in a semi-structured HTML format mixed with other context information. Because of that, such web record information is difficult for computers to understand without proper pre-processing. Given a host webpage and related information needs, how to automatically identify relevant records as well as their internal semantic structures is critical to many online information systems, such as Vertical Search Engines [16] and RSS feeds. The task of extracting records from web pages is usually implemented by programs called wrappers. The process of leaning a wrapper from a group of similar pages is called *wrapper induction* [10, 9, 15, 14, 3, 6, 5, 17, 4, 21, 8]. Due to its high extraction accuracy, wrapper induction is one of the most popular methods to extract web information and it is extensively used by many commercial information systems including major search engines.

However, most traditional wrapper techniques have issues dealing with web records since they are designed to

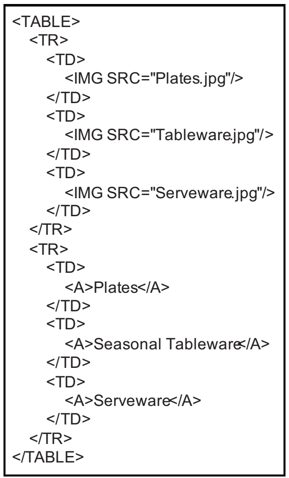
*Work was done when the author was visiting Microsoft Research Asia.

CIKM'09, November 2–6, 2009, Hong Kong, China.
Copyright 2009 ACM 978-1-60558-512-3/09/11 ...$10.00.

extract information from a page, not a record. Specifically, they usually have difficulty in handling the case of *cross records*. For example, Figure 2(a) is a part of a webpage taken from `www.amazon.com` and Figure 2(b) is the corresponding HTML source. In this example, the pictures (`<IMG>`) of three products are presented together in the first row (`<TR>`), while their names (`<A>`) are shown in the second row (`<TR>`). Those three records' HTML sources are crossed with each other. Even though record boundaries are visually clear when such page is rendered for user browsing, as shown in Figure 2(a), there is no clear boundary for partitioning different records from the HTML source. As a result, the lack of record boundaries makes extracting records difficult for most existing wrapper induction methods, especially for those which learn wrappers by inferring repeated patterns from the HTML source. This problem occurs frequently in real-life websites. Almost all image search engines (e.g., Google Image Search) design their search result pages with such layouts. Many shopping websites (e.g., `www.amazon.com`, `www.bestbuy.com`, `www.diamond.com`, `www.costco.com`, etc.) also list some of their products in a similar way as in Figure 2.



(a) Page Layout



```
<TABLE>
  <TR>
    <TD>
      <IMG SRC="Plates.jpg"/>
    </TD>
    <TD>
      <IMG SRC="Tableware.jpg"/>
    </TD>
    <TD>
      <IMG SRC="Serveware.jpg"/>
    </TD>
  </TR>
  <TR>
    <TD>
      <A>Plates</A>
    </TD>
    <TD>
      <A>Seasonal Tableware</A>
    </TD>
    <TD>
      <A>Serveware</A>
    </TD>
  </TR>
</TABLE>
```

(b) HTML Source

Figure 2: Example of Cross Records

Another issue of many page-level wrapper induction methods is the expensive cost of performing tree-mapping for both wrapper induction and data extraction [23]. In these methods, two complete tag-trees need to be aligned with each other, even though most parts of the DOM-tree do not contain user-interested data. Thus, mapping irrelevant information not only wastes lots of runtime, but also interferes with the accuracy of data extraction.

To address the above issues, in this paper, we investigate the wrapper induction technique in record level. We implement a record-level wrapper system and use a novel "broom" structure to represent both records and generated wrappers. With such representations, our system is able to effectively extract records and identify their internal semantics at the same time. In particular, our record-level wrapper technique makes the following contributions:

1. We propose a novel "broom" structure to represent a record, thus our system provides a uniform approach for extracting records from both detail pages and list pages. The record-level wrapper induction approach performs better than the page-level approach and can effectively handle the case of cross records.

2. The record-level approach achieves better efficiency than the page-level approach. The cost of tree-alignment is reduced dramatically by restricting the alignment in the relevant-region of DOM-trees and constructing a wrapper library to avoid duplicated matching.

3. We propose using context words to disambiguate different attributes that are embedded in similar HTML tag trees. Detecting ambiguous attributes and learning context words are fully automatic.

The remainder of this paper is organized as follows. Sec. 2 introduces the fundamental data representation and the flowchart of our wrapper system. Three key technologies of our proposed record-level approach are described in Sec. 3, Sec. 4, and Sec. 5 respectively. Experimental results are presented in Sec. 6. Sec. 7 reviews related work on information extraction. We conclude the paper and discuss our future work in Sec. 8.

2. FUNDAMENTALS

In this section, we first describe the new data structure used to present records and wrappers in our system. Then we present an overview of our record-level wrapper system.

2.1 Data Representation

In our system, a record consists of multiple attributes. For example, a product record can have attributes like "title", "price", "picture", etc. When a page has more than one record, we assign unique IDs ("record id") to them.

Like many other Web information extraction methods, we use the Document Object Model (DOM) [1] to represent an HTML page. In particular, an HTML page is first parsed into a DOM-tree before it can be processed by our system. Nevertheless, our system does not use a full DOM-tree for wrapper induction and data extraction. Instead, we propose a novel data structure, *broom*, to represent a record on a DOM-tree and use such representations for all operations in our system.

As the name implies, a broom has two parts: the "head" and the "stick". The broom head is a record region consisting of sub-trees of a DOM-tree; the broom stick is a tag-path starting from the root tag `HTML` to the top of the record region.

In our system, generated wrappers are also represented in such broom structures. The difference is that special wildcards are introduced in their broom-heads in order to give them more powerful matching ability. (Figure 3)

The reason to include a tag-path in the representation of records and wrappers is two-fold:

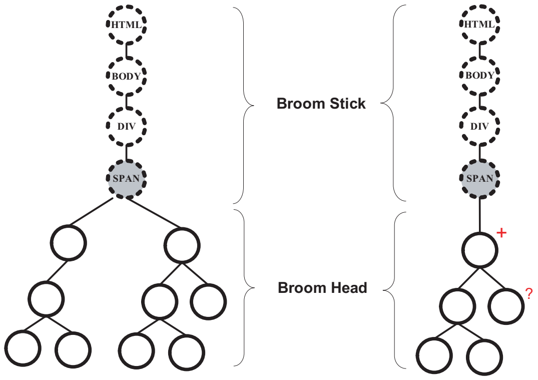


Figure 3: Broom Representation

1. For a specific website, different types of records may have the same sub-tree structure. However, they rarely have the same tag-path at the same time. Normally, a wrapper which is generated for one type of records should not be used to extract other types. Including records' tag-paths makes it much easier to distinguish different types of records by comparing their tag-paths.

2. Records in a website can be grouped by their tag-paths. Then we pose two restrictions to our wrapper system. 1) A wrapper should be learnt only from records which share a same tag-path. 2) A wrapper should be used to only extract records which have the same tag-paths as itself. These two restrictions greatly reduce the computational complexity of our system.

2.2 System Overview

Figure 4 shows the flowchart of our system.

The upper part of this flowchart is the offline training process. First, a set of training pages are converted to DOM-trees by an HTML parser. Then, semantic labels of a specific extraction schema[1] are manually assigned to certain DOM nodes to indicate their semantic functions. Based on these labels, a broom-extraction algorithm can be applied on each DOM-tree to extract records represented by broom structures. Then extracted records are fed to a module to jointly optimize record clustering and wrapper generation [23]. The main output of this process is a set of wrappers.

The lower part of this flowchart is the online extraction process. When a new page enters our system, it is first converted to a DOM-tree. Then, from the wrapper set generated in the training process, one or more wrappers will be automatically selected to align with the DOM-tree. Labels on selected wrappers will be accordingly assigned to the nodes on the DOM-tree. At last, data contained in those mapped nodes will be extracted and saved in an XML file.

Although most of the previous work [2, 19, 13, 21] attempts to conduct fully automatic wrapper induction without labels, labels do matter when wrapper induction plays an important role in a practical system and high extraction accuracy is required. We choose to include an affordable manually labeling process for three reasons: 1) Labels provide more information to distinguish nodes with the same tags. 2) A wrapper with labels explicitly organizes the extracted data into a certain schema. 3) Labeled training data could improve training efficiency by allowing wrapper induction algorithms to focus on the labeled records on a page and ignore the irrelevant parts.

In our system, we use a user-friendly and easy-to-use labeling tool [22] which greatly reduces the effort made in the labeling process. As illustrated in Figure 5, labeling a record can be easily done by several mouse clicks. Those candidate labels shown in the context-menu are loaded from user specified extraction schema. To inform the system that a group of attributes belongs to a same record, users also need to specify a record ID for each attribute they would label. As shown in Figure 5, the record ID selected by the user will appear as prefix in each context-menu item. This helps the user to clearly know which record they are assigning labels to. Once a DOM-tree is labeled, attribute labels and record IDs are all attached to the corresponding DOM-nodes.

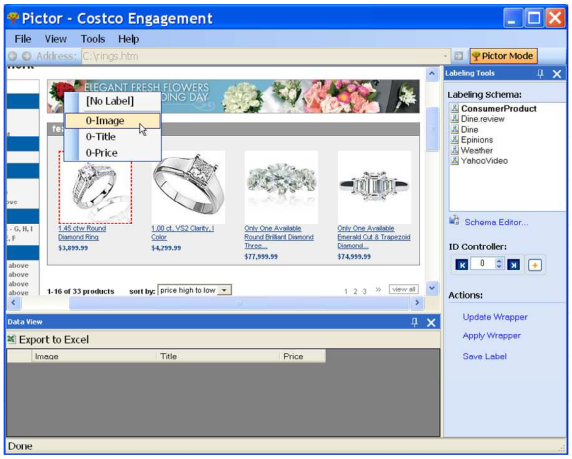


Figure 5: Labeling Tool Interface

3. RECORD WRAPPER INDUCTION

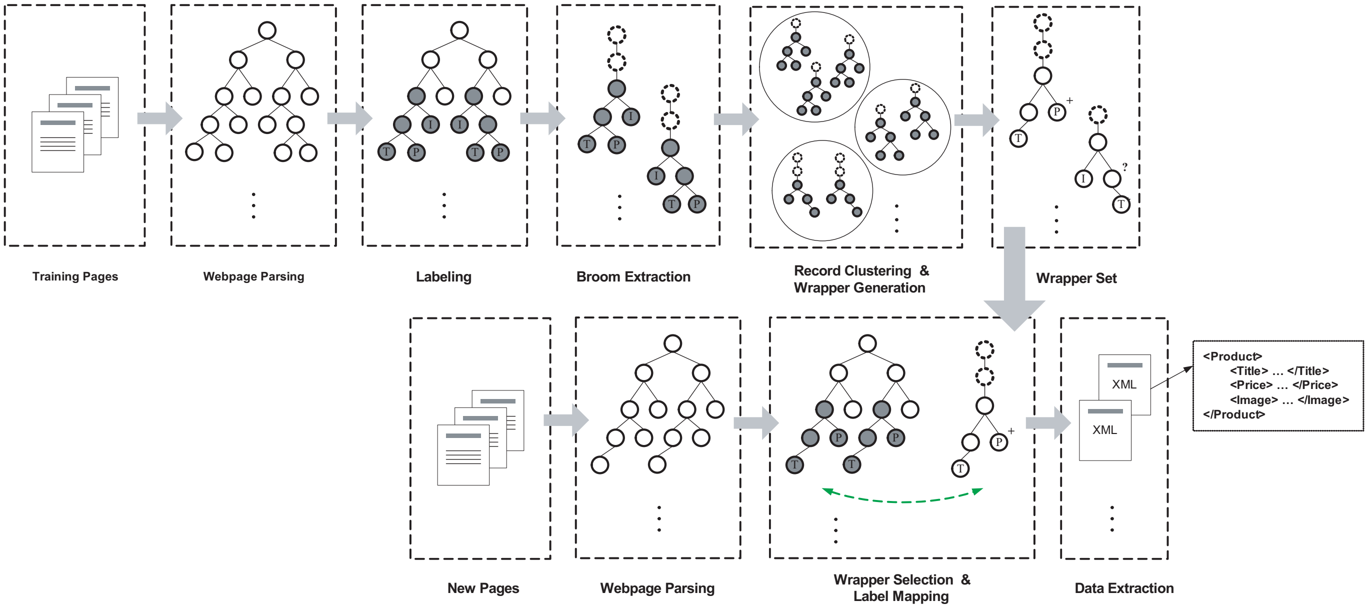
In this section, we describe how to conduct record-level wrapper induction based on broom structures. First, we formally define two important concepts frequently used in our system. Then, we discuss two scenarios encountered in broom extraction process. At last, we walk through the algorithm with an example.

3.1 Minimal Covering Forest & Record Region

Definition 1. *(Boundary Node) Given a labeled DOM-tree and a record ID i, then the boundary node of record i is the root node of a minimal sub-tree which can fully cover all nodes of record i.*

Definition 2. *(Record Region) Given a labeled DOM-tree and a record ID i, then the record region of record i is the smallest set of sub-trees (a forest) which satisfies the following conditions: (1) They can fully cover all nodes of record i (2) They are consecutive siblings rooted at the boundary node of record i.*

[1] *Extraction Schema* refers to the semantic structure of extracted records. It specifies the type of extracted records (e..g, Product) and the attributes in each record.



Note: 1. Dashed DOM-nodes are tag-path nodes; 2. Letters inside circles are attribute labels.

Figure 4: System Overview

In Figure 6, 7, and 8, the gray nodes are boundary nodes and the dotted boxes are record regions for the corresponding labeled records.

Based on the above definitions, the main task of the broom-extraction process is to find the record region (broom head) of each record ID. Once the region is fixed, the corresponding tag-path (broom stick) is also fixed.

3.2 Two Scenarios

Our broom-extraction algorithm is designed to handle two possible scenarios.

Figure 6 illustrates the first scenarios, which is the most common and simple scenario we encountered in real-world data. In this scenario, different records are rooted at different inner nodes (boundary nodes) without overlapping. After locating a boundary node, extracting the record simply equals to copying consecutive child-sub-trees of the boundary node which all contain nodes of the corresponding record ID. As shown in Figure 6, the boundary node is also the last node of the tag-path in an extracted broom.

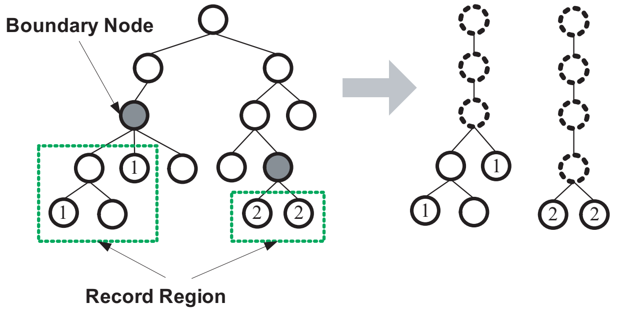


Figure 6: Scenario 1 of Broom Extraction

A special case is shown in Figure 7 where multiple records are rooted at a same boundary node. This case explains why we use a forest instead of a tree to define a record region. In this case, we are unable to find a sub-tree which can fully and only cover one record.

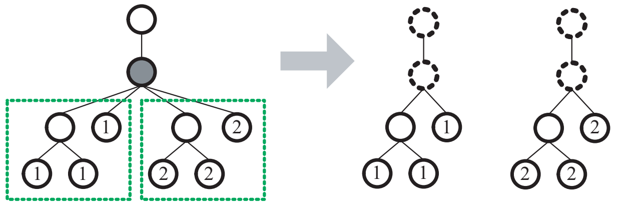


Figure 7: A Special Case of Scenario 1

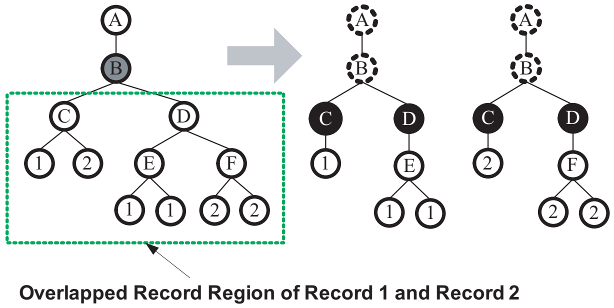
The second scenario is about crossed records. In such scenario, as shown in Figure 8, multiple records can overlap with each other on a DOM-tree. Consequently, their record regions also overlap. When extracting brooms for such records, some inner nodes will be copied to multiple brooms. We call these inner nodes *Non-Exclusive Nodes* (Black node in Figure 8) and the rest DOM-nodes *Exclusive Nodes*. DOM-nodes in the previous scenario are all exclusive nodes.

The property of being exclusive or not will be transferred from extracted brooms to generated wrappers. Exclusive nodes and non-exclusive nodes in a generated wrapper have different matching rules when the wrapper is used in data extraction. See Section 4 for more details.

3.3 Algorithms

The detailed broom-extraction algorithm is listed in Algorithm 1.

Given a labeled DOM-tree d, the extraction routine in Algorithm 1 should be repeated for all record IDs in d and output one broom per record ID. We use the example in



(Black Nodes are non-exclusive nodes)

Figure 8: Scenario 2 of Broom Extraction

```
Algorithm 1 Broom Extraction
Input: Labeled DOM-Tree $d$, Record ID $i$
Output: Extracted broom $b$ of ID $i$
Locate boundary node $n_B$ of ID $i$ in $d$
Get Tag-Path $\mathcal{P}$ starting from root node of $d$ to $n_B$
Output $\mathcal{P}$ to $b$
In all sub-trees rooted at $n_B$, find the first sub-tree $t_F$
  and the last sub-tree $t_L$ that contains record ID $i$
Construct a forest $\mathcal{F}$ starting from $t_F$ to $t_L$
foreach sub-tree $t$ in $\mathcal{F}$
    SelectivelyCopyOneSubtree($t$, $i$, $b$)
end foreach

Method: SelectivelyCopyOneSubtree($t$, $i$, $b$)
Input: Sub-Tree $t$, Record ID $i$, Extracted broom $b$ of ID $i$
if $t$ only contains one record ID $i$ or no record ID then
    copy the whole sub-tree $t$ to $b$
    set all copied nodes as exclusive node
elseif $t$ contains record ID $i$ and other record IDs then
    copy root node $n_R$ of $t$ to $b$
    set copied $n_R$ as non-exclusive node
    foreach child-sub-tree $t_c$ of $t$
        SelectivelyCopyOneSubtree($t_c$, $i$, $b$)
    end foreach
else
    // $t$ only contains record IDs other than $i$
    // do nothing
end if
```

Figure 8 to demonstrate how Algorithm 1 works. Just for the ease of explanation, different inner nodes in Figure 8 are labeled with different letters. Here we only show how to extract a broom b_1 for record 1. Broom extraction for record 2 can be done similarly. First, we need to find a minimal sub-tree which can fully cover nodes of record 1 and the root node B of the minimal sub-tree is the boundary node. Then, the tag path A - B can be fixed and output to the extracted broom. There are two sub-trees rooted at boundary node B. We call SelectivelyCopyOneSubtree for each of them. For the sub-tree rooted at D, since it also contains nodes of record 2, we cannot copy the whole sub-tree to b_1. Instead, we only copy node D and recursively call SelectivelyCopyOneSubtree for sub-trees rooted at E and F. The sub-tree rooted at E only contains nodes of record 1 and can be copied to b_1 as a whole. The sub-tree rooted at F does not contain any node of record 1 and should be ignored. SelectivelyCopyOneSubtree for the sub-tree rooted at C is similarly processed. Then, we get the broom as shown in Figure 8.

After all brooms/records are extracted from the labeled DOM-trees, they will then be fed to a joint optimization process of record clustering and wrapper generating. This process is adapted from our previous work in page level [23]. The work takes mixed DOM-trees for training as input and then combines clustering similar DOM-trees with the same template and generating a wrapper for each cluster in one step. As both template detection and wrapper generation are based on a well-defined pair-wise similarity metrics, that approach can achieve a joint optimization by the criterion of extraction accuracy. To deal with records, we made two main changes:

1. Instead of clustering full DOM-trees, we cluster extracted records represented with broom structures. Consequently, the generated wrappers are also record-level wrappers.

2. In [23], all DOM-trees belonging to a same website will be fed to a single clustering process, whereas, in our system, only records with exact same tag-path will be fed to a common clustering process.

4. RECORD EXTRACTION

In this subsection, we describe how to assemble generated small record wrappers into a wrapper library so that duplicated matching of tag-paths can be collapsed.

4.1 Constructing Wrapper Libraries

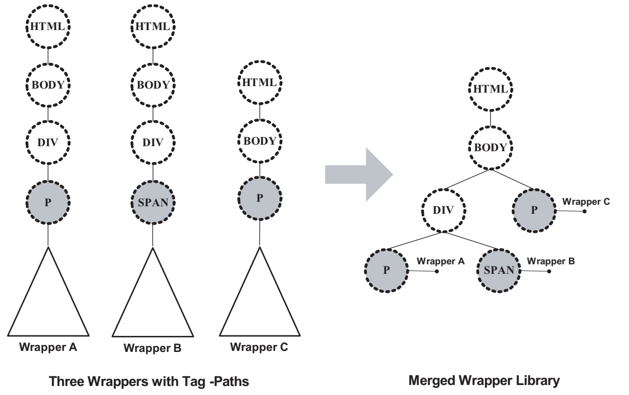
To improve efficiency, wrappers generated from a certain website should only be applied to new pages from the same site. Within this website, each wrapper only attempts to extract records with the same tag-path. Normally, for a potential record which needs to be extracted, only wrappers generated for the same tag-path from the same website can extract it correctly. Therefore, it is wise to impose the above restriction to make the data extraction process more efficient and effective.

In our system, generated wrappers are hierarchically organized according to their original host websites and tag-paths. In particular, we construct a wrapper library for each website. The main task of this construction process is to merge different tag-paths into a tree structure, called wrapper directory, where individual wrappers are linked to the nodes constructed from original boundary nodes (last node of a tag-path). This is a top-down process of merging same prefixes of multiple tag-paths. A simple example is show in Figure 9 to illustrate this process.

Two facts should be stressed for the output wrapper library. (1) Sibling nodes will always have different tags. Otherwise, they should be merged to form a longer common prefix for their tag-paths. (2) Given a tag-path of a wrapper, its location on the library tree is determined. This means wrappers with same tag-path will also be linked at the same boundary node on the library tree.

4.2 Extracting Records with a Wrapper Library

When a new page of a certain website comes to our system, it is first converted to a DOM-tree by an HTML parser; then



(Gray nodes are boundary nodes)

Figure 9: A Simple Example of Constructing a Wrapper Library

the wrapper library generated for this website will be used to perform data extraction from the DOM-tree. If the DOM-tree contains multiple records, they will be assembled as a metadata of the given page. Detailed extraction algorithm is listed in Algorithm 2.

Algorithm 2 Data Extraction with Wrapper Library

```
Input: DOM-Tree $d$, Wrapper Library $\mathcal{L}$
Output: Extracted Record Set $\mathcal{R}$
 1: $root_d$: Root node of $d$
 2: $root_L$: Root node of $\mathcal{L}$
 3: $\mathcal{R}$ = ExtractFromSubDomTree($root_d$, $root_L$)
 4: Return $\mathcal{R}$

Method: ExtractFromSubDomTree($root_d$, $root_L$)
 1: $\mathcal{W}$: Wrapper set linked at $root_L$
 2: $\mathcal{R}$: Extracted record set
 3: $\mathcal{F}$: forest consisting of all sub-trees rooted at $root_d$
 4: if $root_L$ is boundary node then
 5:     while (true)
 6:         search wrapper $w$ in $\mathcal{W}$ with a best match with $\mathcal{F}$
 7:         if $w$ is found then
 8:             Extract a record $r$ from $\mathcal{F}$ with $w$
 9:             $\mathcal{N}$: nodes of $\mathcal{F}$ mapped with $w$'s exclusive nodes
10:             Set all nodes in $\mathcal{N}$ as matched
11:             Add $r$ to $\mathcal{R}$
12:         else
13:             break
14:         end if
15:     end while
16: end if
17: foreach child wrapper library node $child_L$ of $root_L$
18:     foreach child DOM-node $child_d$ of $root_d$
19:         if $child_d$'s tag = $child_L$'s tag then
20:             $\mathcal{R}'$ = ExtractFromSubDomTree($child_d$, $child_L$)
21:             Add all records in $\mathcal{R}'$ to $\mathcal{R}$
22:         end if
23:     end foreach
24: end foreach
```

Note that nodes that have already been matched with non-exclusive nodes of a wrapper will not attempt to match another wrapper. This is to make sure the same record will not be repeatedly extracted.

The algorithm consists of two major steps: tag-path mapping and record extraction. By comparing the tag-path of the target DOM tree with that of the wrapper library, in a top-down manner, only wrappers whose tag-paths can be identified on the target DOM tree will be considered as candidates for record extraction. Once a tag-path of the DOM tree is identified, which means a DOM node $root_d$ is mapped with a boundary node $root_d$ on the wrapper library tree (step 4), then the sub-tree rooted at $root_d$ is considered as the extraction region. All wrappers linked at the $root_d$ will try to match this extraction region. The matching process is basically a top-down tree alignment. The wrapper with a best match will be used to perform the extraction.

Given a DOM tree and a wrapper library, suppose there are t different tag-paths on the DOM tree which can be identified by the wrapper library. The average number of wrappers for each tag-path is k. The average runtime of aligning two trees are A. The runtime of tag-path mapping can be ignored compared to A. Then a rough estimation of the extraction runtime for a given DOM tree is $t(k+1)A$. According to our experiments on various datasets, $t = 1$ for most cases and k is mostly less than five. By applying various programming techniques, we can also control the tree alignment cost A to be 10 - 100 milliseconds. Therefore, the average runtime for extracting records from a page is usually much less than one second.

5. RECORD DISAMBIGUATION

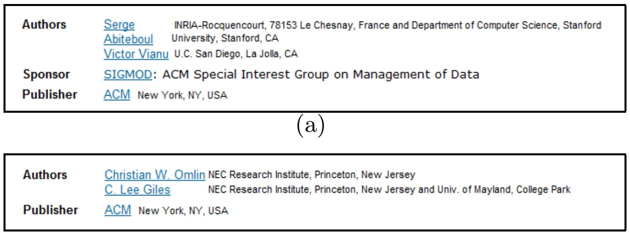
In this section, we describe how to use content text to disambiguate attributes which are embedded in similar HTML tag trees.

By default, we do not consider content text during wrapper induction and data extraction. In wrapper induction process, two wrappers are aligned with each other in order to generate a more general wrapper. During such wrapper aligning, only nodes with same tag and same label can be mapped. Similarly, in data extraction, when a wrapper is aligned with a potential record region of a DOM-tree, only nodes with same tag can be mapped.

Ignoring content does not result in accuracy loss for most cases. The tag-tree structures of records and generated wrappers are already very informative to avoid ambiguity. However, there are still cases where tags are not enough to distinguish different nodes in the aligning process we mentioned above. It can lead to mismatch of different attributes when aligning two tag-trees and thus decrease the extraction accuracy of generated wrappers.

To show this problem, we use an example of two records taken from `portal.acm.org` (Figure 10). The attributes we want to extract for each record are “Authors”, “Sponsor”, and “Publisher”. In this example, the tag-tree structure of the sponsor row and that of the publisher row are identical. Therefore, if we align these two records to infer a wrapper, the publisher row of the second record will be aligned with the sponsor row of the first record because the sponsor row is absent in the second record. Apparently, such mismatch would bring error to the generated wrapper.

To resolve the attribute ambiguity caused by similar tag trees, we propose to utilize content text. In above example, by comparing the content text in the first column, the publisher row of the second record will tend to align with the



Authors Serge Abiteboul INRIA-Rocquencourt, 78153 Le Chesnay, France and Department of Computer Science, Stanford University, Stanford, CA
Victor Vianu U.C. San Diego, La Jolla, CA
Sponsor SIGMOD: ACM Special Interest Group on Management of Data
Publisher ACM New York, NY, USA

(a)

Authors Christian W. Omlin NEC Research Institute, Princeton, New Jersey
C. Lee Giles NEC Research Institute, Princeton, New Jersey and Univ. of Mayland, College Park
Publisher ACM New York, NY, USA

(b)

Figure 10: Examples from portal.acm.org

publisher row of the first record since they both have the same text "Publisher" in the beginning.

Obviously, considering content text will increase the runtime complexity for both wrapper generation and data extraction. We need to compare content text every time we attempt to align two DOM nodes. To limit such complexity increase as well as improve the extraction accuracy, we implement an approach which only considers content text when necessary. First, our approach considers surrounding text in wrapper induction selectively. Only when two consecutive identical tags happen to have different labels, we use text to differentiate them. It is not necessary to waste time on other cases because they are distinguishable by tags. Second, our approach is able to automatically infer template text. When two wrappers are aligned to generate a new wrapper, only overlapped text will be transferred to the new wrapper. In above example, only text in the first column of both records will be transferred to generated wrappers. Third, when a wrapper is aligned with a DOM-tree in data extraction, if there are multiple possible alignments with the same smallest aligning cost, our system will make a decision based on how well the inferred template text is aligned with the text on the DOM-tree. The one with less text mismatch will be chosen as the final solution.

6. EXPERIMENTS

Experimental results are presented in this section to show the performance of our record-level wrapper system. For comparison purpose, we also implement a page-level wrapper induction approach using techniques proposed in [23]. However, we do not compare our system with other wrapper induction systems for the following reasons. First, our system treats DOM nodes as the extraction unit while others treat strings as one. Second, their labeling methods are quite different from ours as well. Therefore, directly utilizing their datasets is impractical.

All experiments were run on a PC, with a 3.06 GHz Pentium 4 processor and 2.0 GB RAM.

6.1 Experiment Settings

6.1.1 Dataset

We collected our experimental data from 16 real-life large-scale websites belonging to four different domains (i.e., online shops, user reviews, digital libraries, search results). Four different extraction schemas are defined for these domains.

There are seven detail page datasets (Table 2) and 11 list page datasets (Table 3). Two websites, i.e. amazon.com and circuitcity.com, provide both list page datasets and detail page datasets. That is why 18 datasets come from 16 websites. Each dataset contains more than 1000 pages. All these pages are manually labeled for the purpose of evaluation.

Table 1: Extraction Schemas

Record	Attributes
Product	title, price, picture, description
Review	author, title, rating, date, comment
Article	title, author, affiliation, year, pages, ...
Search Result	title, snippet, url

6.1.2 Evaluation Metrics

In all experiments, records are equally weighted despite of whether or not they are extracted from a same page. Given a page p for evaluation, we have the extracted metadata $\mathcal{M}_e$ and the manually labeled ground-truth metadata $\mathcal{M}_g$. Both $\mathcal{M}_e$ and $\mathcal{M}_g$ consist of records which belong to page p. The goal of the evaluation is to calculate the *precision*, *recall*, and *F1-value* for $\mathcal{M}_e$ by comparing it with $\mathcal{M}_g$.

First, records in both $\mathcal{M}_e$ and $\mathcal{M}_g$ are sorted according to the order they appear on the DOM-tree of p. Then, we need to align records in M_e with those in M_g because a ground-truth record might be falsely extracted as two records. A dynamic-programming based algorithm is executed to find an optimal alignment which maximizes the overall F1-Value of p.

For each candidate alignment, suppose record r_e in $\mathcal{M}_e$ is aligned with record r_g in $\mathcal{M}_g$, the attribute-level precision (P_{attr}) and recall (R_{attr}) for record r_e can be calculated with the following equations:

$$P_{attr} = \frac{|\text{correctly extracted attributes}|}{|\text{attributes in } r_e|} \qquad (1)$$

$$R_{attr} = \frac{|\text{correctly extracted attributes}|}{|\text{attributes in } r_g|} \qquad (2)$$

Then, we can calculate the average attribute-level precision ($\overline{P_{attr}}$) and recall ($\overline{R_{attr}}$) for all aligned records in $\mathcal{M}_e$. Besides, precision (P_{rec}) and recall (R_{rec}) can also be calculated to show the accuracy in record level:

$$P_{rec} = \frac{|\text{aligned records}|}{|\text{records in } \mathcal{M}_e|} \qquad (3)$$

$$R_{rec} = \frac{|\text{aligned recordss}|}{|\text{records in } \mathcal{M}_g|} \qquad (4)$$

Finally, overall accuracy of $\mathcal{M}_e$ is computed based on averaged attribute-level PR value and record-level PR value:

$$\begin{aligned} precision &= \overline{P_{attr}} \cdot P_{rec} && (5)\\ recall &= \overline{R_{attr}} \cdot R_{rec} && (6)\\ F1\text{-}Vallue &= \frac{2 \cdot precision \cdot recall}{precision + recall} && (7) \end{aligned}$$

6.2 Effectiveness Test

We compare our proposed record-level wrappers with the page-level wrappers proposed in [23] upon all 18 datasets. Results are shown in Table 2 and 3.

The extremely high accuracy indicates that our approach is effective to extract structured data. For list pages, the record-level wrappers improve the page-level wrappers by 7%. For example, Dataset L1 (`www.amazon.com`) contains

300 pages with crossed records and the results proves that record-level wrappers can deal with such scenarios much better than page-level wrappers. For detail pages, the page-level wrappers and the record-level wrappers perform equally well in terms of F1-Value, but the record-level wrappers use less time for all these websites.

Table 2: Results on List Pages (F1-Value)

Website	ID	P-Wrapper	R-Wrapper
www.amazon.com	L1	0.8105	0.9572
www.circuitcity.com	L2	1	1
www.diamond.com	L3	0.9262	0.9926
www.ebags.com	L4	0.9361	0.9962
www.epinions.com	L5	0.9639	0.9701
www.google.com	L6	0.9232	0.9916
scholar.google.com	L7	0.87	1
Average		0.918557143	0.986814286

Table 3: Results on Detail Pages (F1-Value)

Website	ID	F1-Value
www.amazon.com	D1	0.9601
www.buy.com	D2	0.9952
www.circuitcity.com	D3	1
www.compusa.com	D4	1
www.costco.com	D5	1
www.jr.com	D6	1
www.newegg.com	D7	1
www.overstock.com	D8	1
www.target.com	D9	0.9856
www.walmart.com	D10	1
portal.acm.org	D11	0.9292
Average		0.988190909

6.3 Effect of Content Text

Results in Table 2 and 3 are obtained without using the strategy proposed in Section 5. If we also consider the context text, the F1-Value of dataset D11 (`portal.acm.org`) can be increased to 1. As a byproduct, this strategy also results in about 20% increase in extraction time. This is the only website in our dataset which requires content text for disambiguation purpose.

6.4 Efficiency Test

This experiment is designed to compare the average runtime used for extracting a record by page-level wrappers and record-level wrappers. The time for training is limited because it runs upon a small number of pages, while the extraction time is critical for wrapper systems because a huge number of pages will be processed. The 11 detail-page datasets are used for this experiment. Figure 11 shows the results.

Obviously, the runtime difference is significant. On average, record-level wrappers are four times faster than page-level wrappers. It is explainable because aligning two broom structures in record-level wrapper system is much easier than aligning two full DOM-trees in the page-level wrapper system. Also note that, for page-level wrappers, the extraction time notably varies across different websites because the amount of irrelevant information differs in the websites. On the contrary, the extraction time for record-level wrappers has less variation for most websites since the record size is relatively stable.

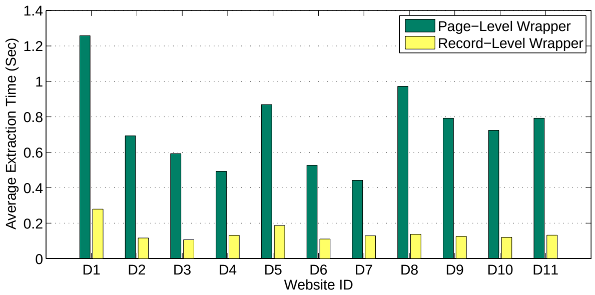


Figure 11: Comparison on Efficiency

In addition, we compare our system with page-level wrapper method regarding labeling effort. We do not directly account the labeling time because it varies a lot with regard to different labelers. Instead, we implement the comparison by counting how many records are used in different methods to train a wrapper set which can achieve a specified accuracy. This somehow shows how much labeling effort can be saved in our system. According to the results, the average record number required by training page-level wrapper is about 129, whereas the number required by training record-level wrapper is about 16. Therefore, our system roughly saves 87% labeling cost compared to the page-level wrapper induction method in [23].

7. RELATED WORK

Our work is in the area of *Web Information Extraction*. It is closely related to previous work on wrapper induction. Several automatic or semi-automatic wrapper learning methods have been proposed. For example, WIEN [10] is the earliest method that we know of on automatic wrapper induction. Other representative work are SoftMeley [9], Stalker [15], RoadRunner [6], EXALG [2], TTAG [4], work in [17], ViNTs [21] and work in [23]. Here, we only discuss TTAG and works in [17] [23] because these three wrapper induction systems also use tree structures to represent webpages and wrappers as in our system.

We refer the reader to two surveys [11, 7] and two tutorials [18, 12] for more work related to information extraction and wrapper induction.

In [17], the tree edit distance is used to measure the distance between two pages. They use traditional hierarchical clustering techniques [20] in which the distance measure is the output of a restricted top-down tree mapping algorithm (RTDM). The RTDM algorithm does not distinguish the node tag and it is designed only for finding the main content in news pages. This restricts that method from being applied to the general information extraction problem. Similar to our system, [17] can also derive a similarity measure between a wrapper (called extraction patterns) and a page when selecting a proper wrapper for extracting data from a new page.

We mention TTAG because wrappers in TTAG are also presented as tree structures with wildcards. They also employ a top-down layer-by-layer alignment, but the alignment in any layer is isolated from that in other layers. As a result, child nodes can still be aligned even when their parent nodes do not match. That is a different strategy from ours.

In [23], a novel wrapper induction system is proposed that expresses a different opinion regarding the relation between template detection and wrapper generation. It takes a miscellaneous training set as input and conducts template detection and wrapper generation in a single step. By the criterion of generated wrappers' extraction accuracy, their approach can achieve a joint optimization of template detection and wrapper generation. A comparison demonstrates that the joint approach significantly outperforms the separated template detection strategy. The idea of joint optimization of wrapper induction and template detection is also considered by our system. The difference is that we did it in record level instead of page level.

Different from all the above wrapper systems are designed to generate wrappers in page level, our systems solves the wrapper induction problem in record level.

8. CONCLUSIONS & FUTURE WORK

This paper describes a record-level wrapper induction system which is able to effectively extract records and identify their internal semantics at the same time. Compared to traditional page-level wrapper methods, the proposed approach not only saves a lot of effort made in manually labeling but also performs data extraction more efficiently. Experimental results on 16 real-life websites from four different domains demonstrate 99% extraction accuracy in terms of F1-Value.

For future work, we would propose to handle the limitation involving tag-paths. In our current version, wrappers can only be applied to extract records with the same tag-paths. In the future, we will see if it is possible to introduce wildcards in tag-paths to enhance the matching power of generated wrappers.

9. REFERENCES

[1] http://www.w3.org/dom/.

[2] A. Arasu and H. Garcia-Molina. Extracting structured data from web pages. In *SIGMOD*, pages 337 – 348.

[3] C.-H. Chang and S.-C. Lui. Iepad: information extraction based on pattern discovery. In *WWW-2001*, pages 681–688.

[4] S.-L. Chuang and J. Y.-j. Hsu. Tree-structured template generation for web pages. In *Proceedings of IEEE/WIC/ACM International Conference on Web Intelligence*, pages 327 – 333, 2004.

[5] W. W. Cohen, M. Hurst, and L. S. Jensen. A flexible learning system for wrapping tables and lists in html documents. In *WWW-2002*, pages 232 – 241.

[6] V. Crescenzi, G. Mecca, and P. Merialdo. Roadrunner: Towards automatic data extraction from large web sites. In *VLDB-2001*, pages 109 – 118.

[7] S. Flesca, G. Manco, E. Masciari, E. Rende, and A. Tagarelli. Web wrapper induction: a brief survey. *AI Communications*, 17:57 – 61, 2004.

[8] A. Hogue and D. Karger. Thresher: automating the unwrapping of semantic content from the world wide web. In *WWW-2005*, pages 86 – 95.

[9] C.-N. Hsu and M.-T. Dung. Generating finite-state transducers for semi-structured data extraction from the web. *Information Systems, Special Issue on Semistructured Data*, 23(8):521–538, 1998.

[10] N. Kushmerick, D. S. Weld, and R. B. Doorenbos. Wrapper induction for information extraction. In *IJCAI-1997*, pages 729–737, 1997.

[11] A. H. F. Laender, B. A. Ribeiro-Neto, A. S. da Silva, and J. S. Teixeira. A brief survey of web data extraction tools. *SIGMOD Record*, 31(2):84–93, 2002.

[12] B. Liu. Web content mining (tutorial). In *WWW-2005*.

[13] B. Liu, R. Grossman, and Y. Zhai. Mining data records in web pages. In *SIGKDD-2003*, pages 601 – 606.

[14] L. Liu, C. Pu, and W. Han. Xwrap: an xml-enabled wrapper construction system for web information sources. In *ICDE-2000*, pages 611–621, 2000.

[15] I. Muslea, S. Minton, and C. Knoblock. A hierarchical approach to wrapper induction. In *Proceedings of the 3rd Annual Conference on Autonomous Agents*, pages 190 – 197, 1999.

[16] Z. Nie, J.-R. Wen, and W.-Y. Ma. Object-level vertical search. In *CIDR-2007*, pages 235–246.

[17] D. C. Reis, P. B. Golgher, A. S. Silva, and A. F. Laender. Automatic web news extraction using tree edit distance. In *WWW-2004*, pages 502 – 511.

[18] S. Sarawagi. Automation in information extraction and data integration (tutorial). In *VLDB-2002*.

[19] J. Wang and F. H. Lochovsky. Data extraction and label assignment for web databases. In *WWW-2003*, pages 187 – 196.

[20] P. Willett. Recent trends in hierarchic document clustering: a critical review. *Information Processing and Management*, 24(5):577–597, 1988.

[21] H. Zhao, W. Meng, Z. Wu, V. Raghavan, and C. Yu. Fully automatic wrapper generation for search engines. In *WWW-2005*, pages 66 – 75.

[22] S. Zheng, M. R. Scott, R. Song, and J.-R. Wen. Pictor: An interactive system for importing data from a website. In *SIGKDD-2008*, 2008.

[23] S. Zheng, R. Song, D. Wu, and J.-R. Wen. Joint optimization of wrapper generation and template detection. In *SIGKDD-2007*.