Schema-Guided Wrapper Maintenance for Web-Data Extraction

Xiaofeng Meng, Dongdong Hu
School of Information
Renmin University of China
Beijing 100872, China
{xfmeng, hudd}@ruc.edu.cn

Chen Li
Information and Computer Science
University of California, Irvine
CA 92697,USA
chenli@ics.uci.edu

ABSTRACT
Extracting data from Web pages using wrappers is a fundamental problem arising in a large variety of applications of vast practical interests. There are two main issues relevant to Web-data extraction, namely wrapper generation and wrapper maintenance. In this paper, we propose a novel schema-guided approach to the problem of automatic wrapper maintenance. It is based on the observation that despite various page changes, many important features of the pages are preserved, such as syntactic patterns, annotations, and hyperlinks of the extracted data items. Our approach uses these preserved features to identify the locations of the desired values in the changed pages, and repair wrappers correspondingly by inducing semantic blocks from the HTML tree. Our intensive experiments on real Web sites show that the proposed approach can effectively maintain wrappers to extract desired data with high accuracies.

Categories and Subject Descriptors
H.2.5 [**Heterogeneous Databases**]

H.2.8 [**Database Applications**]

General Terms
Algorithms, Design, Experimentation

Keywords
Web, extraction, wrapper, maintenance, schema.

1. INTRODUCTION
The World Wide Web has become one of the most important connections of various information sources. A large proportion of the Web data is embedded in HTML documents. The HTML language serves the visual presentation of data in Web browsers, but it is not suitable for automated, computer-assisted information management systems. Thus if data from different sources needs to be integrated, it is necessary to develop special and often complex programs to extract data from Web pages. To achieve this goal, people have developed wrappers [7], which are specialized programs that can automatically extract data from Web pages and convert the information into a structured format. Different methods [1, 2, 5, 7, 8, 9, 13, 14, 15, 17, 18, 19] have been proposed to automate the wrapper-generation process.

There are many challenges in constructing wrappers. Often, a wrapper needs to be developed for each data source because of the heterogeneous page structures from different Web sites. Thus generating wrappers for different sources could be time consuming and error prone. At many Web sites, pages representing similar contents often have different structures. For instance, the news pages at "Yahoo News" [24] contain several different structures, while they have similar semantic contents and data items. Meanwhile, Web pages can be very dynamic and continually evolving, which results in frequent changes in their structures. As a consequence, earlier constructed wrappers may stop working. It is often critical to update existing wrappers so that they can still extract the desired data. One way to deal with such situations is to rewrite wrappers from scratch using the new pages. Clearly this method is inefficient due to the heavy workload to the system developers.

Recently, several methods are presented to address the problem of automatically extracting Web data and repairing (maintaining) wrappers. Kushmerick et al. [10, 11] define a problem called "wrapper verification," which checks if a wrapper stops extracting correct data. Their proposed solution analyzes pages and extracted information, and detects the page changes. If the pages have changed, the designer is notified, so that she can re-learn the wrapper from the pages with the new structure. Knoblock et al. [8] develop a method for repairing wrappers in the case of small mark-up changes. Chidlovskii [4] presents an automatic-maintenance approach to repairing wrappers under the assumption that there are only small changes. Crescenzi et al. [18] and Arasu et al. [19] propose an approach to automatic data extraction by automatically inducing the underlying template of some sample pages with the same structures from data-intensive Web sites.

In this paper, we propose a novel schema-guided approach to wrapper maintenance, called SG-WRAM. It is based on our previous work of a schema-guided wrapper generator SG-WRAP [14, 15]. The maintenance solution is based on the following observations. Although changes of HTML documents are various, some features of the extracted data items in these pages are often preserved, such as syntactic features, hyperlinks, and annotations (see Section 3.1). It is feasible to recognize data items in the changed pages using these features. In addition, the schema for the

WIDM'03, November 7-8, 2003, New Orleans, Louisiana, USA.

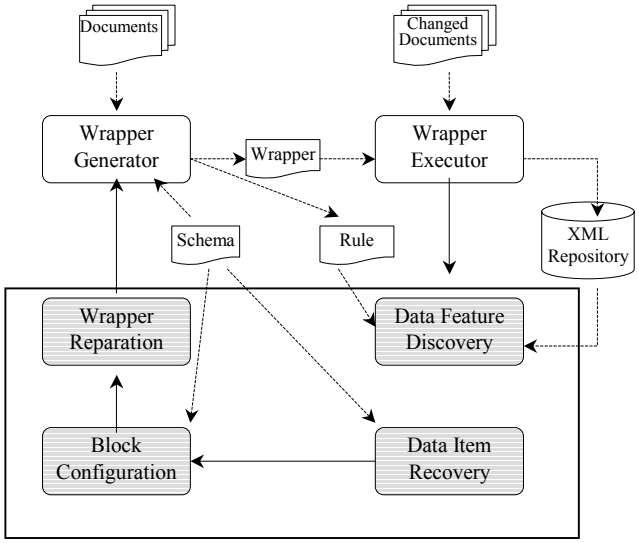


Figure 1: The Architecture of SG-WRAM

extracted data does not change. Specifically, we maintain wrappers in the following four steps. At first, features are obtained from the user-defined schema, the previous extraction rule, and the extracted results. Secondly, we recognize the data items in the changed page with these features. The third step groups the items according to the schema. Each group, called *semantic block,* is a possible instance of the given schema. Finally, the representative instances are selected to re-induce the extraction rule for the new page. Our experience with real Web pages shows that this approach can deal with not only simple changes, but also many complex changes including context shifts, structural shifts [4], and their combinations.

The rest of this paper is organized as follows. Section 2 gives an overview of our SG-WRAM system, and the preliminaries about our schema-guided wrapper generator SG-WRAP. In Section 3 we present the detail about our approach to wrapper maintenance in the case of page changes. Section 4 reports our intensive experiments on real Web sites. Section 5 discusses related work. In Section 6 we conclude the paper.

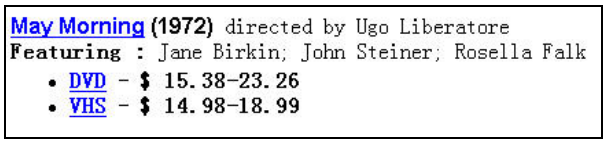
2. SG-WRAM: A Wrapper-Maintenance System

Figure 1 depicts the architecture of the implemented system [16]. The system consists of three major components: a wrapper generator, a wrapper engine, and a wrapper maintainer.

The *wrapper generator* provides a GUI that allows users to provide an HTML document and an XML schema (DTD), and specify mappings between them. Then the system will generate an extraction rule (wrapper) for this page. The rule extracts data from the page and constructs an XML document conforming to the specified XML schema [14, 15].

The *wrapper engine* provides the execution environment of the generated rules. Given an extraction rule and an HTML page, the engine runs the rule to extract the target data from the page. In the case where the rule fails, the engine informs the wrapper maintainer to fix the rule.

The *wrapper maintainer* automatically repairs a wrapper that fails to extract correct data after the pages have changed. In this



(a) Original page

(b) Changed page

Figure 2: Sample Web Pages

paper we focus on the wrapper-maintenance problem. Wrapper-verification techniques of testing whether a wrapper stops working are represented in [10].

2.1 Schema-Guided Wrapper Generation

The visual-supervised-wrapper-generation approach serves as the base for our techniques of automatic wrapper maintenance. The main idea of the approach is the following. A user defines the structure of her target information by providing an XML schema in the form of a DTD. For instance, Figure 3 shows such a DTD for the HTML page shown in Figure 2(a). The user can define this schema to indicate what she wants to extract from the pages and what the output result XML document should look like. Given an HTML page, by using a GUI toolkit, the user creates mappings from useful values in the HTML page to the corresponding schema elements. Internally the system parses the HTML page into a DOM tree [21], and computes the corresponding internal mappings from the HTML tree to the schema tree. Using these mappings the system can generate a tree pattern and output an extraction rule as an XQuery expression.

```
<!ELEMENT VideoList (Video+)>
<!ELEMENT Video (Name, Director, Actors, Price)>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Director (#PCDATA)>
<!ELEMENT Actors (#PCDATA)>
<!ELEMENT Price (VHSPrice, DVDPrice)>
<!ELEMENT VHSPrice (#PCDATA)>
<!ELEMENT DVDPrice (#PCDATA)>
```

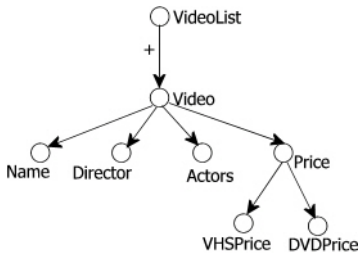


Figure 3: Target DTD schema

Annotations of data values are used to maintain wrappers in our approach. An *annotation* of a value is a piece of descriptive information that can describe the semantic meaning of the content of this value. The annotation may lie in another text node near this value in the HTML tree, or lie in the same text node of this value. Our experience with many Web pages shows that the node containing the annotation always appears before the node of data

Table 1: Annotations for HTML data values

Data values in HTML page	Annotations
May Morning	-
Ugo Liberatore	directed by
Jane Birkin; John Steiner; Rosella Falk	Featuring
15.38-23.26	DVD
14.98-18.99	VHS

value in the depth-first traversal (DFS) order of the tree. For instance, Table 1 shows the annotations of several data values in the page shown in Figure 2(a). Note that the annotation of a value could be empty (e.g., "May Morning"). In addition, it may not be the nearest node to the data value node. For instance, the annotation "VHS" of the price value "14.98-18.99" is not the closest node.

2.2 Inducing Extraction Rule

Each rule in SG-WRAM is an FLWR XQuery expression [23], which, if applied on the HTML page, can generate an XML document conforming to the DTD. In general, in such an extraction rule:

- A schema element marked with a symbol "+" or "*" (e.g., `VideoList`) corresponds to a clause of "`FOR … RETURN …`".
- Any other element (e.g., `Name`, `Director`, etc.) corresponds to a clause of "`LET … RETURN …`".

The structure of the rule is based on the DTD schema. For each `LET` or `FOR` clause, the system fills in the appropriate XPath expression in the HTML tree based on the internal mappings. The following is an instance of internal mappings for Figure 2(a).

MAPPING(D: "Ugo Liberatore",
HP: ……/text()[0][contains(null, "directed by")],
SP: VideoList/Video/Director).

Symbol *D* ("Data") is the data value, *HP* ("HTML Path") is the path to this value in the HTML tree, and *SP* ("Schema Path") shows the path to its corresponding DTD element. Note that the XPath function of "contains()" in *HP* records the annotation for this data value. The first parameter of this function is the path to the annotation starting from the data value, and the second is the value of the annotation.

The rule-induction algorithm used in SG-WRAP traverses the DTD tree from the root element. The algorithm first finds all the mappings whose *SPs* contain this element. Then it computes a common path of the sub-trees in the HTML tree that can include these mappings. Next, if the element is marked by a symbol "*" or "+" in the schema tree, there may be multiple sub-trees that contain the input mappings of the current instance. Thus the algorithm searches similar sub-trees by trying to generalize the paths. Finally, the generalized common path is used to generate the rule for this element. If the element is not a leaf, the algorithm calls itself recursively for each of its child elements. The rules returned are added to the current rule as subrules. After all these steps, the extraction rule is created.

3. Maintaining Extraction Rules

Web pages may change from time to time, and the extraction rules could stop working due to these changes. Some slight changes in a Web page layout could prevent the wrapper from extracting data correctly. Thus we want to repair the extraction rules automatically so that they can work for the new pages.

Table 2: Data Features

ID	DTD Element	L (hyperlink)	A (annotation)	P (data pattern)
1	Name	True	NULL	[A-Z][a-z]{0,}
2	Director	False	Directed by	[A-Z][a-z]{0,}
3	Actors	False	Featuring	[A-Z][a-z]{0,}(.)*
4	VHSPrice	False	VHS	[$][0-9]{0,}[0-9](.)[0-9]{2}
5	DVDPrice	False	DVD	[$][0-9]{0,}[0-9](.)[0-9]{2}

The problem of wrapper maintenance includes two sub-problems, namely wrapper verification and wrapper reparation. Much work has been conducted on wrapper verification [8, 10, 11, 12]. We focus on wrapper reparation in this paper. Our approach is based on the observations that many important features of the pages are preserved after the changes, such as syntactic features, annotations, and hyperlinks. In addition, the schema of the extracted data does not change. Our approach of wrapper maintenance has four steps.

- *Data-feature discovery*: Data features are computed from the given DTD, the previous extraction rule, and the previous extracted results.
- *Data-item recovery*: Data features are used to recognize the relevant data items in the new page.
- *Block configuration*: We group the recognized data items according to the user-defined schema and the HTML tree structure. We compute a semantic block as an instance of the given schema.
- *Rule re-induction*: The computed instances are used to re-induce the new extraction rule for this changed page.

In this section we discuss these four steps in details. We use the pages in Figure 2 to illustrate these steps. It shows part of the original page (a) and the changed page (b) obtained from the Yahoo Shopping site [25]. The original extraction rule fails to extract correct data from the new page because the page has changed its underlying template.

3.1 Data-Feature Discovery

In this step we compute various features from the data values. A *syntactic feature* includes certain syntactic characteristics of data items, such as data patterns, string lengths, and so on. Here we focus on data patterns for simplicity. For instance, a street address often has the data pattern consisting of a street number and a street name. Data patterns can be represented as regular expressions. An *annotation feature* of a data item is a descriptive string in the HTML page for this value. A *hyperlink feature* of a data item shows whether the data item has an associated hyperlink.

Table 2 shows features for the DTD elements in our running example. It has a row for each element with a unique ID. We consider three important features of each DTD element, represented as a triple (*L*, *A*, *P*):

- *L*: A Boolean value (TRUE or FALSE) to indicate whether the data item *d* corresponding to this element has an associated hyperlink or not.
- *A*: An annotation of the data value *d*.
- *P*: A data pattern, represented as a regular expression for this data value.

The system computes the entries for each DTD element as follows. The value of hyperlink feature *L* is straightforward: its value is TRUE if there is a hyperlink associated to the corresponding data

element in the HTML page, and FALSE otherwise. The annotation feature *A* is recorded in the extraction rule with the XPath function of "*contains()*"(see Section 3). The data pattern feature *P* of each DTD element is generated it by studying several example pages with the same structure [6], for instance, using machine learning techniques or applying pattern-extraction techniques [3].

3.2 Data-Item Recovery

In this step, we traverse the new HTML tree following the depth-first traversal (DFS) order. For each leaf node *n*, if the data value could be an annotation of an item, we try to find the corresponding value of this item starting from the annotation node. Otherwise, we check the item table to see if it satisfies the three conditions (features) of an item row *r:* (1) if *r.L* = "TRUE" ("FALSE"), then node *n* does (not) have an associated hyperlink. (2) The annotation of node *n* should be the same as *r.A* (it can be null). (3) The string of this value is recognized by the regular expression (data pattern) of *r*. If all these conditions are satisfied, the leaf node *n* is called an *instance* of the item *r*, and node *n* is expected to be an instance of the corresponding DTD element. For example, if a node string *n* is an instance of the Item 2 in Table 2, then *n* should have an annotation "Directed by," have no hyperlink, and be accepted by the data pattern "[A-Z][a-z]{0,}". In this case, this node is one of the target nodes of Item 2. In this step, we create an item-instance list stored in an array.

In the current version of algorithm, if the annotation of an item changed in the new pages, it could be treated as a different item. For instance, suppose "Our Price: $1.00" changed to "Price: $1.00", the same data value "$1.00" has different annotations: "Our Price" and "Price". But it's difficult to judge if they correspond to the same DTD element by other features. In this case, the system could inform the user about these changes, and let the user decide if the annotations play the same role in the HTML documents.

Meanwhile, we remove possible noises that can repeatedly occur in most of the instances. For example, the pages about "E-book" from www.amazon.com contains a sentence of "*Click here for more info.*" This sentence could be recognized as a possible data item in the step of item recovery. We remove this kind of noises from the item instance table by computing the frequencies of occurring of these data values. The method relies on the fact that if the value of an item in the user defined schema can repeatedly occur in the pages, they will often have a more precise data pattern. For the limited space, we'll not introduce the details in this paper.

At last, we get the item instance list as shown in Table 3. The ID here is the identifier of an item in Table 2. Here we only provide the results for the first three items.

Table 3: Item Instance Table

No.	ID	PATH
1	1	…**table[1]**/tr[0] /td[1]/span[0]/b[0]/a[0]/text()[0]
2	2	…**table[1]**/tr[0]/ /td[1]/span[1]/text[contains(/preceding-sibling::b[0],"Directed by")]
3	3	…**table[1]**/tr[0]/ /td[1]/span[2]/text()[contains(/preceding-sibling::b[0],"Featuring")]
4	1	…**table[2]**/tr[0] /td[1]/span[0]/b[0]/a[0]/text()[0]
5	2	…**table[2]**/tr[0]/ /td[1]/span[1]/text[contains(/preceding-sibling::b[0],"Directed by")]
6	3	…**table[2]**/tr[0]/ /td[1]/span[2]/text()[contains(/preceding-sibling::b[0],"Featuring")]

3.3 Block Configuration

After recovering all the possible data items, we want to find out the underlying structure of these data items. In other words, we compute the nodes in the HTML tree that contain all the target data. These data items are often grouped in different *semantic blocks*. An HTML document can be viewed as containing a set of semantic blocks, and each semantic block is a fragment of the HTML tree that conforms to the user-defined schema. A semantic block includes instances of schema elements. It could be a sub-tree, or several sibling sub-trees that include all the instances of the schema elements. Each instance is a row of the Item Instance Table (Table 3). In this step we construct semantic blocks from the new HTML page.

A semantic block is called *atomic* if it satisfies the following conditions:

- It is a sub-tree or set of sibling sub-trees; and
- The occurrences of data values conform to the definition of the schema.

Intuitively, an atomic semantic block is the minimum extractable unit from which we can extract an XML document conforming to the schema. A *match* between a semantic block *A* and the schema can be one of the following three cases:

- *Over match*: There is at least one item *i* in the schema that occurs at least twice in block *A*.
- *Full match*: Block *A* contains all items of the schema and satisfies the constraint of each item in the schema, such as '+' or '*', '?' etc.
- *Partial match*: Block *A* contains a proper subset of items of the schema.

In the block-configuration step, we first identify the level of block configuration in a top-down manner. At the level *k* of the new HTML tree, each sub-tree is viewed as a possible block. Since most Web pages containing interesting data come from a common underlying template, most of the data items are always located in a "big" sub-tree or several sub-trees, although there is often a complex structure in these sub-trees. We compute the *weight* of a sub-tree, which is the number of possible data items in this sub-tree. We use the sub-tree weight for excluding some low-weighted noisy sub-trees.

After classifying all possible blocks at a level, the number of blocks in each kind of matches is counted without considering the non-important sub-trees. The possible blocks are then divided into 3 groups: full-match group, partial-match group and over-match group. The group *R* with the largest number is used to decide the next step.

- If *R* is the full-match group, return all the blocks that are fully matched.
- If *R* is the partial-match group, merge sub-trees at the parent level *k*-1.
- If *R* is the over-match group, turn to the child level *k*+1 and continue the block-configuration step.

If the blocks in the changed pages can fully match with the schema, this step will stop at the level where the system finds all the fully matched blocks. In many cases, we cannot find the right level to get all the fully matched blocks. For instance, the changed page contains only parts of the data items from the original page. To solve this problem, at a certain step the system finds that data items are scattered in several partially matched blocks, while they

should be in the same block. So we should merge them. First, we merge two sibling sub-trees. If the result is still a partial match, we continue merging sibling sub-trees with the same parent. We repeat this step until the algorithm stops at a level (say, *n*), where we find that the algorithm gets too many over-matched blocks at level *n-1* and gets too many partial-matched blocks at level *n+1*. Thus in the Item Instance Table (Table 3), the items from row 1 to row 3 are finally decided in the same block and those from row 4 to row 6 are in another block.

3.4 Rule Re-induction

The results of the block-configuration block are a set of semantic blocks. In the rule re-induction step, we choose a representative block as an instance to construct mappings from the data values in the new HTML page to the DTD schema. Then we can re-induce the new extraction rule by calling the wrapper generator.

In our running example, we choose the first block in the Table 3 to construct the internal mappings as follows:

M1'(*D*: "Lucky Day",
HP:…/**table**/tr[0]/td[1]/span[0]/b[0]/a[0]/text()[0],
SP: VideoList/Video/Name)
M2'(*D*: "Penelope Buitenhuis",
HP: …/**table**/tr[0] /td[1]/span/text()[contains(/preceding-sibling::b[0],"Directed by")],
SP: VideoList/Video/Director),
M3'(*D*: "Amanda Donohoe, Tony Lo Bianco, Andrew Gillies",
HP:…/**table**/tr[0]/td[1]/span/text()[contains(/preceding-sibling::b[0],"Featuring")],
SP: VideoList/Video/ Actors),
……

By running the rule induction algorithm of SG-WRAP system [14,15], we generate the new extraction rule.

In this process it is possible for the algorithm to select a bad instance, making some of the data items not successfully extracted. In addition, while applying the repaired wrapper to other pages with the similar structure, some data items may not be extracted. For instance, some of the pages returned from the same search engine could have some small structural differences from other pages, making a few data items not extractable by the wrapper generated from the other pages. To solve this problem, our SG-WRAM system can automatically select other representative instances from the results of the block configuration step to refine the repaired wrapper. Since the extraction rule uses an XQuery expression, it is easy to integrate two extraction rules as follows:

- If the rule from new instances contains new predicates for a certain data item, the predicates are added into the extraction rule.
- If the new rule contains a new path to a data item, the path is added into the extraction rule.

4. Experiments

To evaluate the effectiveness of our approach to wrapper maintenance, we conducted intensive experiments on real Web pages. We monitored 16 Web sites from October 2002 to May 2003. All of them are data-intensive Web sites [18]. For each Web site, we periodically archived some pages of the same URL or pages from the same query to the site's search engine. In order to increase the chance to get changed pages from these sites, we collected several sets of pages with different structures from each site. We built different wrappers for these different structures. For instance, at www.amazon.com, pages from the site's book search engine and pages of best sellers have different structures, thus they had different wrappers. For each set of collected pages we did the following:

1) Ran the SG-WRAP system on the earliest pages and generated a wrapper for each set of pages.
2) Applied the initial wrapper to the newly collected Web pages. By manually checking how many data items corresponding to the elements in the DTD can be correctly extracted, we made sure if a page had changed.
3) For each set of changed pages, we obtained a repaired wrapper using our SG-WRAM system. The new wrapper was applied on the changed pages.

Table 4: Initial wrapper on changed pages

Name	Recall (R%)	Precision (P%)	Item Number	# of changed pages
1Bookstreet Book	82.54	100	6	12
Allbooks4less Book	0	-	4	15
Amazon Book (search)	40.49	100	6	15
Amazon Magazine	20.01	100	5	15
Barnesandnoble Book	0	100	5	15
CIA Factbook	0	100	10	5
CNN Currency	50.00	100	6	15
Excite Currency	42.86	100	11	18
Hotels Hotel	0	-	4	15
Yahoo Shopping Video	0	-	6	15
Yahoo Quotes	0	-	6	10
Yahoo People Email	0	-	3	10

4.1 Evaluation Metrics

We define the following assistant parameters:

- *CN*: number of correct data items that should be extracted in a page;
- *EN*: number of data items extracted by the wrappers;
- *CEN*: number of the correct data items extracted by the wrappers.

We use two metrics, *Precision* and *Recall,* to evaluate the results of our algorithm of wrapper maintenance.

- *Recall (R)*: proportion of the correctly extracted data items of all the data items that should be extracted. It can be presented as "*R = CEN/CN*".
- *Precision (P)*: proportion of the correctly extracted data items of all the data items that have been extracted. It can be presented as "*P = CEN/EN*".

4.2 Analysis of Changed Pages

After applying the initial wrappers on the newly collected pages, the system found those wrappers with low recalls or precisions. We manually checked if the pages had format changes. The results are shown in Table 4. The first column lists the wrapper name. The next two columns list the value of recall (*R)*

and precision (*P*). Symbol "-" means that the value of *P* was not computable. It is because the initial wrappers cannot get any data

Table 5: Wrapper maintenance

Web site	R%(IR)	P%(IR)	R%(EX)	P%(EX)
1Bookstreet Book	98.67	71.26	100	100
Allbooks4less Book	75	32.69	75	51.34
Amazon Book (search)	83.05	36.3	83.05	90.74
Amazon Magazine	100	60.15	100	100
Barnesandnoble	78.72	43.13	78.72	100
CIA Factbook	100	100	100	100
CNN Currency	100	100	100	100
Excite Currency	100	100	100	100
Hotels Hotel	50	35.61	50	41.87
Yahoo Shopping	100	51.49	100	92.86
Yahoo Quotes	100	100	100	100
Yahoo People	100	53.54	100	100

item from the changed pages, thus the value of EN and CEN are all 0. The forth column provides the number of correct data items of each set of pages considered in this experiment. The last column shows the number of changed pages at the source.

The initial wrappers for Web sites `1Bookstreet`, `Amazon Book`, `Amazon Magazine`, and `Excite Currency` could still extract data items from the changed pages. The Web site `CIA Factbook` changed its page structure and still kept its non-table structure. Other sites also changed their page template, making their initial wrapper invalid. For instance, the site `Yahoo Quotes` changed its page structure from a complex table to a non-table structure.

4.3 Effectiveness Results

After identifying those changed pages, the system used the four steps to maintain the wrapper. The new extraction rule was re-induced from a single changed page. The new rule was only refined using the instances within the page, without being refined using other Web pages.

4.3.1 Basic Results

For each wrapper, we computed the metrics shown in Table 5. We monitored the results on two stages, after the second step of item recovery (IR), and after the system applying the re-induced extraction rule on the changed pages (EX). R%(IR) and P%(IR) represent the metrics recall and precision of the former stage, R%(EX) and P%(EX) show the corresponding metrics of the later one.

Note that the values of R%(IR) and R%(EX) are always the same except the first example, it is because after the steps of block configuration and rule re-induction, the recognized data items during item recovery are still covered by the re-induced extraction rule. While we found the value of "P%(IR)" was lower than the value of "P%(EX)" for almost all the examples, as we have explained before, the later steps, especially the step of block configuration, have excluded many of the noisy sub-trees.

In the results, `CIA Factbook`, `CNN Currency`, `Excite Currency`, `Yahoo Quote` achieved perfect results on the four metrics. `CIA Factbook` benefited from the fact that all the data features of its date items were preserved, although the Web site had changed its underlying template completely. The data features include the data pattern, annotation, and hyperlink.

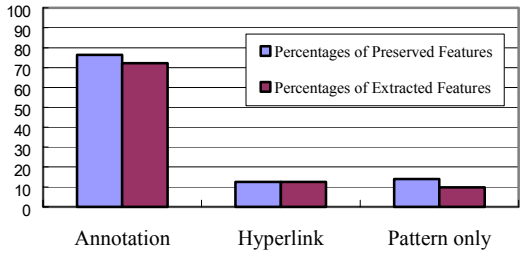


Figure 4: Discussion on Data Features

As to `CNN Currency`, `Excite Currency`, `Yahoo Quote`, all of them had all the data items in a simple table, and all their data items were purely numeric. In addition, their annotations were perfectly preserved; e.g., the table head was the same. So the system could precisely find out all the data items without any noises in the steps.

For the site `1Bookstreet`, all its data features were preserved, so the item "You Save" with a small structure change could be located in the changed pages. Because the item of "Availability" has multiple expressions at the sites, and the example pages did not contain all the patterns of these expressions, several corresponding item instances were missed when we took the item recovery step. But during the final step of rule re-induction, the common path computed in the extraction rule helped find these item instances. Thus the metric of "R%(EX)" was 100% in the extracted results.

For the site `Amazon Book`, its annotation of an item "Our Price" changed to "Buy New," and the system treated them as different data items. So the repaired extraction rule did not contain this data item. Meanwhile, the pages contained another big sub-tree showing "the most popular books," which made the system compute three un-wanted blocks from them. Thus the P%(EX) metric failed to get a satisfactory value. Similarly, for the site `Barnesandnoble`, the system failed to recognize the data item of "Availability" because of the changed data pattern.

For the sites `Hotels` and `Allbooks4less`, both of them got poor results on almost all the metrics. Although most of the data patterns of them did not change, almost all the other features (annotation and hyperlink) were not preserved. So during the item recovery step, too many noises could not been effectively eliminated in the latter steps. The system could hardly find correct blocks from the results of block configuration, and the repaired extraction rule could not work effectively.

As these examples cover many representative structural changes, the experiments showed that our maintenance algorithm has high practicability. Meanwhile, the experiments also show that the usage of annotation and hyperlink features has more advantages for wrapper maintenance than using syntactic features (data pattern) only.

4.3.2 Extensive Discussion on Data Features

Figure 4 shows how data features were preserved in the changed pages and how effectively they were extracted by our approach. We selected a few sample pages from each Web site and get statistics.

We find that about 80% of the data items had annotations in the original pages. The repaired wrapper could extract almost all of these data items by using the data pattern and annotations. This result means that most of the annotations were preserved after the various Web page changes and can still been extracted from them. On the other hand, although fewer data items had the data hyperlink feature preserved, most of them corresponded to the item "Name" of a book or other merchandise. But the features of hyperlinks are nearly perfectly preserved in all the changed sample pages. Note that a data item can have both of these two features, so in Figure 4, the sum of the percentage of the three features is slightly larger than 100%.

The "Pattern Only" bars in Figure 4 illustrate the situation where the data items do not contain the other two features. The statistics tell that there's only a small quantity of data items have the feature of syntactic features (pattern) only, and among them, a less number of data items lost their sole features in the changed pages. Although it's a relatively small part of all the data items, it often makes the wrapper maintenance much harder by bringing too much noises during the step of item recovery, and additionally increasing the difficulties of the next steps of block configuration and rule re-induction. In fact, as to the examples of `Hotels` in Table 5, the system had to recognize three out of four data items by having to use data patterns only, which produced too many noises and greatly affect the following steps (see Table 5).

5. Related Work

There are two aspects on wrapper maintenance: *change detection* and *wrapper reparation*. Kushmerick [10] presented an algorithm called RAPTURE for solving wrapper verification. RAPTURE compares the pre-verified label with the label output by the wrapper being verified. Specifically, it compares the value of various numeric features of the strings comprising the label output by the wrapper. Then, an overall probability is computed about whether the wrapper is correct for the page. If the overall probability is less than a user-defined threshold, the page is considered to be unchanged; otherwise, it is considered to have changed. If the generic features of some data fields are similar to the pre-verified one, their system cannot detect the change.

Lerman and Minton [12] addressed the issues of both change detection and wrapper reparation. For the first problem, they applied machine learning techniques to learn a set of patterns that can describe the information being extracted from each of the relevant fields. They used the starting and ending strings as the description of the data field. If the pattern describes statistically the same proportion of the test examples as the training examples, the wrapper is considered to be still correct. Next, the wrapper re-induction algorithm takes a set of training examples and a set of pages from the same source, and uses machine learning techniques to identify examples of the data field on the new pages. This method could produce too many candidates of data fields, many of which could be noises. The process of clustering the candidates for each data field does not consider the relationship of all data fields (schema). Furthermore, the top ranked cluster may not include all correct candidates. If we only use the examples in top ranked cluster for the re-induction, the new rule may not be fit for other pages.

The above approaches have the following limitations:

1) Both approaches heavily rely on the syntactic features of data items, making the step of identifying data items possibly confused. This limitation also makes them not be able to detect some changes in which the syntactic feature does not change. For example, suppose the *Title*, *List Price*, and *Our Price* are to be extracted from the following original table.

Title	List Price	Our Price
Data on Web	$29.00	$23.00
Java Programming	$59.00	$49.00

The following is the new table.

Title	Our Price	List Price
Data on Web	$23.00	$29.00
Java Programming	$49.00	$59.00

The approaches in [10, 12] cannot detect this kind of changes, because the generic features and data patterns of *List Price* and *Our Price* are the same. Instead, our work considers annotation features, thus when applying the extraction rule, our approach will find that the annotations correspond to the paths that have changed and find that the page has changed.

2) Our algorithm of block configuration conforming to the user schema can effectively find the appropriate level in the HTML tree where the correct semantic blocks reside, while the approach of using top ranked clusters in [12] cannot guarantee the semantic integrality and correctness.

There are also some works on automatic wrapper generation. For instance, ROADRUNNER [18] compares HTML pages belonging to the same "page class" to generate a wrapper based on their similarities and differences (mismatches). The approach focuses on generalizing the wrapper to eliminate the mismatches of tag structure to induce the template for a set of Web pages. The approach in [19] computes "equivalence classes" from sample pages and discovers sets of tokens associated with the same type constructor in the template to create the input pages. Then the template is deduced by using these sets. Both of these approaches have the advantages of not requiring any pre-defined input. However, their extracted data may not be annotated. Although [20] proposed a method annotating data items by the word in the template, the problem still has not been resolved satisfactorily. Another problem of these approaches is their high time complexity, especially when the web pages are complicated.

These automatic approaches provide an alternative way to deal with the wrapper maintenance problem. In some cases where there's no pre-defined input, it will be a good choice to use the approach of automatic extraction. If the pages to be extracted contain similar semantics but there're varies of structures; or there exists plenty of needed pre-input features, the maintenance approach may take more advantages.

6. Conclusion

In this paper, we proposed the novel schema-guided approach to wrapper maintenance, called SG-WRAM. The approach utilizes the effective data features (such at syntactic features, hyperlink, and annotation of extracted data items) that are often preserved after page changes. They are used to identify the locations of the desired data items in the changed pages. Semantic blocks conforming to the user-defined schema are used for grouping data items to recognize the underlying structure of Web pages

effectively and precisely. Our intensive experiments with real Web pages showed that our proposed approach can effective maintain wrapper in the presence of page changes.

Acknowledgements

This research was partially supported by the grants from 863 High Technology Foundation of China under grant number 2002AA116030, the Natural Science Foundation of China (NSFC) under grant number 60073014, 60273018, the Key Project of Chinese Ministry of Education (No.03044) and the Excellent Young Teachers Program of M0E，P.R.C (EYTP).

REFERENCES

[1] Ashish N, Knoblock C A. Wrapper generation for semi-structured Internet sources. SIGMOD Record, 1997, 26(4): 8-15.

[2] Baumgartner R, Flesca S, Gottlob G..Visual Web Information Extraction with Lixto. In Proceedings of the VLDB 2001, 119-128.

[3] Brin S. Extracting patterns and relations from the world wide Web. In International WebDB Workshop, Valencia, Spain, pages 172-183, 1998.

[4] Chidlovskii B. Automatic repairing of Web Wrappers. In 3rd International Workshop on Web Information and Data Management, 2001, 24-30.

[5] Doorenbos R, Etsionoi O, Weld D S. A scalable comparison-shopping agent for the world-wide-Web. In Proceedings of the First International Conference on Autonomous Agents, 1997, 39-48.

[6] Gupta A., Harinarayan V., Quass D., and Rajaraman A. Method and apparatus for structuring the querying and interpretation of semistructured information. United States Patent number 5,826,258, 1998.

[7] Hammer J, Brenning M, Garcia-Molina H, Nestorov S, VassalosV, Yerneni R. Template-based wrappers in the TSIMMIS system. In Proceedings of ACM SIGMOD Conference, 1997, 532-535.

[8] Knoblock C A, Lerman K, Minton S, Muslea I. Accurately and Reliably Extracting Data from the Web: A Machine Learning Approach. Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, 2000, 23(4): 33-41.

[9] Kushmerick N, Weil D, Doorenbos R. Wrapper induction for information extraction. In Proceedings of International Joint Conference on Artificial Intelligence (IJCAI), 1997, 729-735.

[10] Kushmerick N. Regression testing for wrapper maintenance. In Proceedings of AAAI, 1999,74-79

[11] Kushmerick N. Wrapper verification. World Wide Web Journal, 2000, 3(2): 79-94.

[12] Lerman K. and Minton S.. Learning the common structure of data. In AAAI2000.

[13] Liu L, Pu C, Han W. XWRAP: An XML-enabled Wrapper Construction System for Web Information Sources. In Proceedings of ICDE, 2000, 611-621.

[14] Meng X F, Lu H J, Wang H Y, Gu M Z. SG-WRAP: A Schema-Guided Wrapper Generator. Demonstration in Proceedings of ICDE, 2002, 331-332.

[15] Meng X F, Lu H J, Wang H Y, Gu M Z. Schema-Guided Data Extraction from the Web. Journal of Computer Science and Technology (JCST), 2002,17(4).

[16] Meng X F, Wang H Y, Hu D D, SG-WRAM: Schema Guided Wrapper Maintenance, Demonstration in Proceedings of ICDE, 2003, 750-752.

[17] Sahuguet A, Azavant F. Building Light-Weight Wrappers for Legacy Web Data-Sources Using W4F. In Proceedings of VLDB, 1999, 738-741.

[18] Crescenzi V, Mecca G, Merialdo P. ROADRUNNER: Towards Automatic Data Extraction from Large Web Sites. In Proceedings of VLDB, 2001, 109-118.

[19] Arasu A, Garcia-Molina H. Extracting Structured Data from Web Pages. In Proceedings of ACM SIGMOD Conference, 2003, 337-348.

[20] Arlotta L, Crescenzi V, Mecca G, Merialdo. Automatic Annotation of Data Extracted from Large Web Sites. In WebDB, 2003.

[21] W3C. DOM Document Object Model (DOM) Level 2 Core Specification， http://www.w3.org/TR/DOM-Level-2-Core

[22] W3C. XML Path Language (XPath) 2.0, http://www.w3.org/TR/xpath20/

[23] W3C. XQuery 1.0: An XML Query Language, http://www.w3.org/TR/xquery/

[24] Yahoo News: http://news.yahoo.com

[25] Yahoo Shopping: http://shopping.yahoo.com