Automatic Data Extraction from Data-rich Web Pages

Dongdong Hu and Xiaofeng Meng

School of Information
Renmin University of China
{hudd,xfmeng}@ruc.edu.cn

Abstract. Extracting data from web pages using wrappers is a fundamental problem arising in a large variety of applications of vast practical interests. In this paper, we propose a novel technique to the problem of differentiating roles of data items from Web pages, which is one of the key problems in our automatic extraction approach. The problem is resolved at various levels: semantic blocks, sections and data items, and several approaches are proposed to effectively identify the mapping between data items having the same role. Intensive experiments on real web sites show that the proposed technique can effectively help extracting desired data with high accuracies in most of the cases.

1   Introduction

The World Wide Web has become one of the most important connections of various information sources. A large proportion of data on the web is embedded in various HTML documents. The HTML language serves the visual presentation of data in Web browsers, while it is not suitable for automated, computer-assisted information management systems. This is not expected to change soon, even when XML is more and more popular today. 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*, which are specialized programs that can automatically extract data from web pages and convert the information into a structured format.

There have been many works on semi-automatic and manual data extraction [1] [2] [3] [4] [5], in the past years. These approaches require human interactions to build sample mappings between the output results (or schemas) and items in the HTML pages, after that extraction rules will be induced for extracting pages having similar structures. Besides the flexibility of these approaches, there are still many challenges in constructing such wrappers. Firstly, the user should have good knowledge of contents in the web pages and should try to select more proper samples to cover more possible situations; also, another important problem is how to maintain existing wrappers if the corresponding web pages take changes on their layouts.

There still exists several other automatic approaches, Exalg [6] and Roadrunner [7], which can automatically extract data from *data-intensive* [7] web sites. Pages from data-intensive sites are created through encoding values, from the underground database, into web pages using templates [6]. In other words, such pages usually have the same schema and accordingly similar structures. Either Roadrunner or Exalg use

the structure information of sample pages to induce HTML tags-based templates (see Section 5). A template covers the constant parts of the HTML sources' string sequence, and the left parts which are variant are viewed as the right data to be extracted.
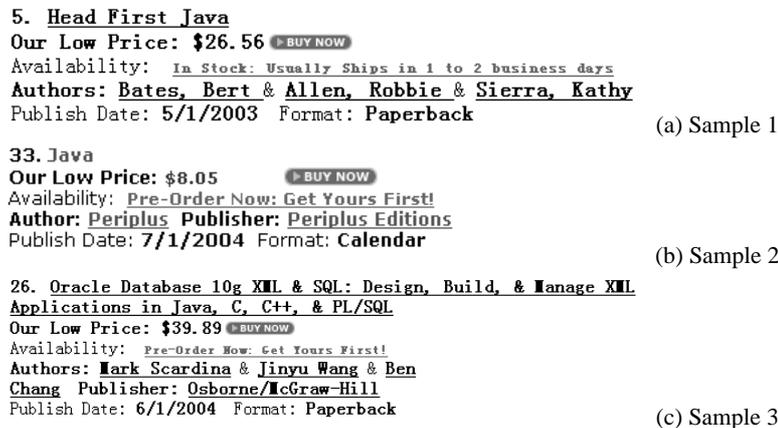

(a) Sample 1

(b) Sample 2

(c) Sample 3

Fig. 1. Sample HTML fragments from Buy

Unlike the previous work described above, we propose a novel automatic approach to generate wrappers without any user interactions. Firstly we break up the problem of automatically constructing a wrapper into four subproblems:

1. Discovering blocks (called *semantic block* in this paper, which corresponds to an instance of the schema) containing data to be extracted from sample pages;
2. Differentiating roles of data items in the semantic blocks;
3. Inducing schema describing the contents in the pages;
4. Computing extraction rule and generating wrappers.

Among these four steps, a simple way to tackle problem (1) is to use a technique of tree comparing for discovering data-rich sections [8]. For example, in Fig. 1 are three extracted blocks from sample pages from www.buy.com. Problem (3) is introduced in [9], and problem (4) is widely discussed in almost all the previous work of data extraction. In the previous work [3] we proposed an XQuery [10] based extraction rule system. Our work focuses on problem (2), which serves as the key step for the extraction problem. Once we have got a set of semantic blocks, which can be viewed as a set of instances of the schema, which describes the contents in the page, to be induced in the next step. To induce the schema from the instances, we firstly differentiate the roles of data items in them. In other words, for a data item in a sematic block, we want to find out the data items taking the same role, corresponding to the same schema element, in all other semantic blocks. For example, referring to Fig. 1, the items of "Head First Java" and "Thinking in Java with CDROM" are both corresponded to the schema element `Title`. The process of differentiating roles are named *Identification* in this paper. After all the data items are identified, each one is assigned a label indicating

its role. Data items having the same roles have the same labels. Note that the label here does not provide any semantic information, but a symbol representing the role of a data item.

This paper is structured as follows. In the next section the challenges of we faces and some preliminary knowledge is described, in section 3 we introduce the technique of simple identification. Section 4 discusses the complex identification, and section 5 compares our work to the related work. In section 6 we report the experimental results. Finally in section 7 we highlight the conclusion and future research direction.

2 Challenges and Our Approach

The HTML pages are parsed into a tree representation in our work. Considering the DOM [11] model, a non-leaf node corresponds to a pair of HTML tags, such as "`<tr>...</tr>`". Nodes are ordered according to their occurrence sequence in the HTML page. Each leaf node is either of type PCDATA, or a single tag such as "`<tr>`". All the values are contained in the text nodes, and can be uniquely defined to be an element in the tree with an XPath [12] expression.

An important characteristic of pages belonging to the same site and encoding data of the same schema, is that the data encoding is done in a consistent manner across the pages. For example, in Fig. 1 each of the three fragments represents a record from database and is a *semantic block* conforming to the same schema {Book{Title, Price, Availability, Author|Authors{Person+}, Publisher?, PubTime?, Format}}. Intuitively a semantic block of a schema can be viewed as an instance of the corresponding schema in the web pages. In the semantic blocks are a set of data items without being identified their roles. For example, from Fig. 1(a) we get the following data items: "Head First Java", "Our Low Price:", "$26.56", "Availability:", ..., "Paperback". Here some text nodes are not included, e.g. text nodes containing the sequence number of the record, and text nodes containing only the symbol '&'.

As we saw in Fig. 1, the underlying schema for the contents of pages contains three kinds of special situations: iterations, optionals and disjunctions. To induce a schema which can precisely describe the data in the pages, we should be able to detect all these three situations when identifying data items.

Iterations means that the instances of these schema elements can iteratively appear in the page. Iterations are the most common cases in the real web pages, e.g. the element `Person` in the schema for Fig. 1.

Optionals indicate the cases that given a schema element, we cannot find corresponding data values in every semantic blocks. Optionals are represented by the cardinality "?". For instance, the element `Publisher` in in the schema describing Fig. 1 is optional.

Disjunctions can be represented by "|", e.g. "$a|b$", which means at the corresponding relative position can be "a" or "b", but only one of them. An example for disjunction is in Figure 1, which used `Author` for single author, while `Authors` for multi-authors. As to the corresponding schema, there can be a schema element of `Author|Authors`.

3   Simple Identification via HTML Path

The simple identification handles the situation that the pages have relatively regular layouts. From the point of tree structure, the pages that simple identification faces have the common feature that all the data items of the same role have the same HTML paths(in XPath [12] expressions), which start from the roots of the subtrees containing the semantic blocks and end at the nodes of the data items. Based on this characteristic, the following rule are used for identification data items in such pages.

Rule of HTML Path: Two data items with the same HTML path inside the semantic blocks have the same role.

Our experiments reveal that pages from most of web sites can be simply identified by this rule. For instance, pages from CDplus (http://www.cdplus.com, see Fig. 2) can be perfectly identified using this rule, each semantic block of which takes a row in the table and the data items having the same role locate in the same column, correspondingly with the same HTML path inside the semantic block.

Beach Boys	**Today!/Summer Days (And Summer** (CD)	3/20/2001	Cdn$18.49 (US$13.70)
Beatles	**A Hard Days Night** (CD)	1/1/1990	Cdn$18.99 (US$14.07)
Benet, Eric	**A Day In The Life** (CD)	4/27/1999	Cdn$17.49 (US$12.96)
Bet.E And Stef	**Day By Day** (CD)	10/1/2002	Cdn$18.49 (US$13.70)

Fig. 2. Fragment of sample pages from CDPlus.com

4   Complex Identification

Some situations that cannot be handled by simple identification, e.g. items with same role but different HTML paths, or different roles but with the same HTML path, are resolved in complex identification. The complex identification tries to divide the semantic blocks into smaller cells for improving the reliability of identification, thus the identification can be conducted inside matched cells without having to deal with the entire semantic blocks. This guarantees that the identification will not completely fail if the blocks can be divided.

Firstly we give the definition of *Occurrence Path*. An occurrence-path [6] of a data item is the path from the root to the certain node in the HTML tree. The only difference between an HTML path and a Occurrence path is that the former has predicates in the path expression. Our experiments show that data items having different occurrence paths usually have various roles in most of the complex pages. Two neighboring data items are divided into two *path-groups* if they do not have the same occurrence path. For instance, the three co-authors are assigned in one path-group for their common occurrence paths //table/tr/td/#text.

4.1 Block Segmentation Using Template Items

Based on the observation that web pages are designed to be consumed by human users, and therefore they usually contain text strings, i.e. annotations (labels) [14] [15], whose goal is to explicate to the final user the intentional meaning of the published data [13]. These kinds of strings are used to divide the semantic blocks into small cells in our approach.

Discovering Possible Template Items The items in the semantic blocks can be classified into two categories: (*i*) Template items, which may be contents of the template of the pages, e.g. annotations, or some invariant strings; (*ii*) Data values, which are the exact values we want to extract from web pages.

The following features are used to find template items from the semantic blocks:

1. Template items of the same role have the same value in all the semantic blocks if occurring.
2. Template items of the same role have the same occurrence paths inside the semantic blocks.
3. The syntactic features, mainly the patterns of the template items in the semantic blocks are usually similar. For instance, all the annotations in 1 start with a capital letter and end with a colon.

The algorithm for searching possible template items starts from one of the blocks, and scans all other semantic blocks for discovering candidates by checking the above features. Meanwhile, all the discovered candidates are sorted by the order they occur in the semantic blocks. Turning to the example pages in Fig. 1, we get seven possible template items with their occurring order: "`Our Low Price:`", "`Availability:`", "`Author:`", "`Authors:`", "`Publisher:`", "`Publish Date:`", "`Format:`". We also get a list of template items from each semantic block. Note that in this step it is not necessary to precisely find out all the template items. The key purpose of the possible template items are used to divide semantic blocks into smaller sections for future operations.

Dividing Semantic Blocks into Sections Using Template Items Since the discovered template items are expected to repeatedly occur in the semantic blocks. They can be used as markers to divide the semantic blocks to smaller sections. Furthermore, from the point of visual effect, sections between common template items' pair provide similar information.

Before dividing the real semantic blocks, we firstly build a *full-division* on all the possible template items. The full-division confirms that each template item may have a section containing data items on each of both sides. Fig. 3 shows the full-division of the example in Fig. 1, in which each section is assigned an ID using the sequence number.

Thus each semantic block is divided by comparing the list of discovered template items with the full-division (suppose the function $Id(section)$ gets the ID of a section).

1. For each pair of neighboring template items t_i and t_{i+1} in a semantic block, we first find out all the sections $\{s_j, ..., s_k\}$ contained between t_i and t_{i+1} in the full-division. After that, the ID of the section s' between t_i and t_{i+1} is assigned to be $(Id(s_j)\text{-}Id(s_k))$;

2. As to the section s' just before the first template item t_0 in the semantic blocks, we first find the section s_j just before t_0 in the full-division, then the ID of s_0 is assigned to be $(0\text{-}Id(s_j))$; conversely the last section is handled.

Consequently the relation between a section s' in a semantic block and a section s in the full-division can be one of the follows: (*i*) If $Id(s) = Id(s')$, the two sections have the same role. (*ii*) $Id(s)$ is included by $Id(s')$, which means that $Id(s)$ locates at the range specified by the starting ID and ending ID of $Id(s')$, we know that there exists situation of optionals. In other words, at least one of sections included in the full division do not appear in the corresponding semantic block to be divided. Other, if the intersection of the ranges of two sections' ID is not empty, the two sections are called matched. And there exists at least one pair of data items having the same role in the two matched sections.

Considering our running example, the results of this step are shown in Fig. 4(a). For instance, the section between "Availability:" and "Authors:" is assigned to be 2-3 because the corresponding starting section's ID is 2 and the ending section's ID is 3 in the full-division. The ID containment here indicates that the semantic block does not contain one of the sections next to the template item "Author:" in the full-division.

```
{0} Our Low Price: {1} Availability: {2} Author: {3}
Authors: {4} Publisher: {5} Publish Date: {6} Format: {7}
```

Fig. 3. The full-division

```
{0} Our Low Price: {1} Availability: {2-3} Authors: {4-5}
Publish Date: {6} Format: {7}
```

(a) Buy sample 1

```
{0} Our Low Price: {1} Availability: {2} Author: {3-5}
Publish Date: {6} Format: {7}
```

(b) Buy sample 2

```
{0} Our Low Price: {1} Availability: {2-3} Authors: {4}
Publisher: {5} Publish Date: {6} Format: {7}
```

(c) Buy sample 3

Fig. 4. Semantic blocks divided into sections

4.2 Mapping Data Items in Matched Sections

Computing Similarities between Data Items To identify the data items in the matched sections, several additional rules are employed besides the rule of HTML path. All the rules are considered under the condition that the mappings to be verified are between matched sections.

Rule of Occurrence-Path: (*i*) Two items from *different semantic blocks* with different occurrence paths usually have different roles; (*ii*) Two items in the *same section* with different occurrence paths usually have different roles;

In practice, this rule is proved to be true in most of the situations. For example, the three co-authors in Fig. 1(a) have the same occurrence path //table/tr/td/#text and have the same role. An extension to this rule is that *consecutive* items with the same occurrence paths may possibly have the same role. In other words, there may be iterators, e.g. the example of three co-authors in Fig. 1(a). The rule may fail in several special cases, e.g. we report an exception in the sample pages of `uefa(teams)` in Section 7.2.

Rule of Visual Information: Two data items having the same role usually have the same visual information.

The visual information is also very important to the extraction problem. In real pages, data items having same role usually have the same appearances to keeping consistence on visual effects. In our work, we consider the following visual information: (i) Whether the data items use the same font and have the same font size; (ii) Whether the data items use the same color in the pages. (iii) Whether the data items have hyperlinks on them.

Rule of Context: Data items with incompatible contexts have different roles.

This rule captures the fact that the contents in the semantic blocks conforming to the same schema, which guarantees that data items of a role always occurs at the relatively same position comparing with other data items. The contexts of two items are said compatible if the contexts of one item can be contained by the other item's. For instance, items in unmatched sections always have different roles.

Rule of Syntactic Feature: Data items having the same role usually have same syntactic features.

This rule is based on the fact that items of the same role often have the same syntactic features [16] [14], e.g. the data items "In Stock: Usually Ships in 1 to 2 business days" and "Pre-Order Now: Get Yours First" in Fig. 1 can both be defined by the regular expression "[A-Z]([\\w]|[\\s])* (:)([\\w]|[\\s])*" indicating that each item starts with a capital letter and contains a colon inside the strings. More details about computing regular expressions of data items can refer to [16].

To judge if two data items, d_1 and d_2, matches, we compute an aggregate similarity of them, denoted as $Sim(d_1, d_2)$, based on all the four above rules.

$$\lambda_1 * RO(d_1, d_2) + \lambda_2 * RC(d_1, d_2) + \lambda_3 * RS(d_1, d_2) + \lambda_4 * RV(d_1, d_2). \quad (1)$$

Here the functions RO, RC, RS and RV correspond to the four rules presented above. For each of them, it takes value 1 if the two data items satisfy the certain rule, else its value is 0. $\lambda_i (i = 1, 2, 3, 4)$ is the weight of each rule which shows the importance of the rule in the process. For simplicity, the values of all of them are 0.25 (The values of them can be configured for better representing the real situations when extracting).

Thus if the similarity are larger than a pre-given threshold, we know that the data items match most of the above rules and are considered to have the same role. Moreover, the aggregate similarity greatly decreases the chance that one of the rule's failure resulting an incorrect identification.

Detecting Iterations, Optionals and Disjunctions *Iterators* inside a path-group is detected by computing the similarity of all the data items in it. For example, suppose there're two matched path-groups $p = \{d_1, d_2, d_3\}$ and $p' = \{d'_1, d'_2\}$, with d_1 matches d'_1 and d_2 matches d'_2. Meanwhile, if the similarity $Sim(d_1, d_2, d_3)$ and similarity $Sim(d'_1, d'_2)$ are larger than the pre-given threshold, we say that there can be iterations inside p and p', and the two existing matchings are transformed into an extended matching that $\{d_1, d_2, d_3\}$ matches $\{d'_1, d'_2\}$. Note that we take into account here only the possible iterators inside path-groups. In fact, if a page contains more than one semantic blocks, it's also an iterator at the granularity of blocks. On other hand, separators are also used to discover possible internal iterations, e.g. the symbol '&' in Fig. 1 can be used as separator for discovering internal iterations.

The *optionals* can be naturally discovered. (*i*) At the granularity of sections in semantic blocks, optionals are detected by the ID containment judging (see Section 4.2). (*ii*) At the granularity of path groups and data items, the situation is that the data items having the same role do not appear in all the compatible sections.

The problem of detecting *disjunctions* can be transformed into the problem of discovering optionals under the environments of web pages. For instance, suppose we find a case of disjunction in a page, which can be expressed as $(a?|b?)$ or $(a|b)$. Since the page itself has determined that only one or zero of a and b can appear in a semantic block. The situation can also be described in $(a?, b?)$ without losing correctness.

Consequently, the complex identification problem is achieved through differentiating roles of sections, and then differentiating roles of path-groups and data items inside the set of matched path-groups. The results of this step are that each data item in the semantic blocks is assigned a label, may not have semantic meaning, which corresponds to a element in the schema describing the contents of the pages.

5   Related Work

There have been lots of work on data extraction [17]. These work can be classified by the degree of needs of human interaction (manual, semi-automatic and automatic approaches), sources of information targeted (human made v.s. machine generated), etc. In this paper, we mainly focus on the works on automatic approaches: Exalg [6] and Roadrunner [7].

Both Exalg and Roadrunner use the page creation model of encoding values into web pages using templates. The Roadrunner approach starts from the entire first input pages as the initial template. Then, for each subsequent sample page, it checks if the page can be generated by the current template, otherwise, it modifies the current template using *mismatch* technique to ensure that the updated template can generate all the pages seen so far. The Exalg approach works in two stages. In the first stage, it discovers "equivalence classes", sets of tokens associated with the same type constructor in the

(unknown) template for creating the input sample pages. In the second stage, it uses the above sets to deduce the template by continuously growing the LFEQs (for Large and Frequently occurring EQuivalence classes) using several heuristic rules.

The key features differ our work from them are the followings:

1. Both Exalg and Roadrunner focus on inducing tag-based template from the input HTML tag sequences. While the problem of tag-based template is that too less template tokens may making it unable to precisely locate the data items encoded in them. For example , Exalg fails to extract the 4 attributes in the template "$<${Name:$*<$br$>$,(Email:$*<$br$>$)?,(Organization: $*<$br$>$)?,(Update:$*<$br$>$)?$>$" for the reason that this template contains too less, just two template tokens, associated with each type constructors. This kinds of problems can be perfectly resolved by dividing semantic blocks into sections using template items in our approach.

2. Both Exalg and Roadrunner treat the input HTML document as a token sequence without considering the characteristic of tree structure of HTML document. They also ignore the following features, e.g. visual information, syntactic features, for assisting the extraction problem.

3. Roadrunner assumes that the "grammar" of the template used to generate the pages is union-free, which means that it cannot deal with pages with disjunctions in the page schema. Moreover, the complicated heuristic search involving "backtracking" for dealing the situation that the current template does not generate an input page makes Roadrunner difficult to low the complexity of the algorithm, since the search is exponential in the size of the schema of the pages.

On the other hand, some work related to wrapper maintenance [16] [14] [15] also take the step of locating data items in the changed web pages for repairing broken wrappers. While these were under the direction of pre-defined schema or pre-acquired features of data items, e.g. pre-computed syntactic features, pre-computed extraction rule, etc.

6 Experiments

Based on the techniques of item identification above, we have developed a prototype system. Several intensive experiments have been conducted on collections of real web pages. All the experiments have been conducted on a machine with an Intel Pentium IV processor working at 2GHz, with 256 MBytes of RAM, running Windows 2000 Server and Sun JDK 1.4. For each collection, the experiment contains the following steps:

1. Manually build a schema for presenting the data values to be extracted from the pages. The schema has nothing to do with the process of extraction, but only for verifying the extracted results.

2. Run the system to extract data values from the pages. Firstly the system automatically induces an wrapper from exactly *two* sample pages, and then the wrapper is applied on other pages in the collection.

3. Manually check the results.

Table 1. Experimental results of our collections

No.	source	#n	#a	#c	#p	#i	#b	#t(ms)	extr.	#size(K)
1	amazon(hotel)	10	3	3	0	0	20	17.36	yes	86.5
2	buy(bag)	10	5	5	0	0	50	73.15	yes	241
3	buy(book)	10	8	8	0	0	50	86.16	yes	231
4	cdplus	10	5	5	0	0	40	12	yes	88.8
5	cnn(search)	10	3	3	0	0	30	28.93	yes	61.2
6	ebay(buy it now)	10	6	6	0	0	100	50	yes	241
7	ebay(auction)	10	6	5	0	1	100	58.29	yes	233
8	ecampus	10	8	8	0	0	20	28.89	yes	66.9
9	hotels	3	11	11	0	0	50	121.65	yes	476
10	yahoo(people)	10	3	3	0	0	20	19	yes	96.4
11	yahoo(shopping)	10	6	4	2	0	20	21.82	yes	130

The web pages for experiments were collected from two means: (*i*) Pages collected by ourselves from the various well-known site (see Table 1); (*ii*) Pages ever used in the related works (see Table 2). The pages collected by ourselves are universally larger in size and more complex structures than the pages from the related works.

6.1 Evaluation Metrics

Based on the manually built page schema, we evaluate the effectiveness of our approach using the three cases of *Correct*, *Partially-Correct* and *Incorrect* for each schema element. (i) *Correct*: All the data values corresponding to the schema element are correctly extracted and identified; (ii) *Partially Correct*: Only parts of the data values matching the schema element are extracted from the pages; (iii) *Incorrect*: Otherwise, the schema element is classified as incorrect.

6.2 Effectiveness Results

Table 1 shows the results on the pages collected by ourselves, and Table 2 provides the results on the pages from the related works. Both tables contain the following elements: (*i*) *source*: short description of each collection, *#n*: the number of sample pages; (*ii*) *#a*: the number of elements in the manually built schema; *#c*, *#p* and *#i* showing the cases of *correct*, *partially-correct* and *incorrect*; (*iii*) *#b* the number of semantic blocks in the two sample pages for creating wrapper; *#t* is the total time needed to inducing schema from the pages, starting from finding semantic blocks and ending after inducing the schema, and *#size* is the total size of the two sample pages used for inducing wrapper; (*iv*) finally *extr.* conclude that if the collection can be automatically extracted.

As it can be seen from the table, for most of the collections, the system was able to correctly induce a schema using only very short time. Let us describe some more details: Firstly, there're totally 2 reports of failing to induce a correct wrapper from the sampling. We conclude the reasons into two aspects: (*i*) The example of `Barn.&Nob.(sw)` fails because the system fails to discover the semantic blocks. Thus even the system can

correctly extract one data items from each semantic block, it still fails to successfully built a correct wrapper. (*ii*) The failure on `tennis` can due to that the we can hardly get a good pair of sample pages for inducing wrapper. The pages in the collection do not have a consistent schema, making the system fail to induce a correct schema, and accordingly a correct wrapper. Thus the induced wrapper can only extract randomly some data items. In fact, Exalg [6] also extract only parts of the data items.

Table 2. Experimental results of collections from related works

No.	source	#n	#a	#c	#p	#i	#b	#t(ms)	extr.	#size(K)
1	amazon(cars)	21	3	3	0	0	11	17.5	yes	53.1
2	Barn.&Nob.(sw)	10	3	1	2	0	39	33.55	no?	77
3	buy(prod)	10	8	8	0	0	2	5.33	yes	44.1
4	MLB(player)	10	4	4	0	0	141	65.5	yes	77.9
5	rpmfind(by dist.)	20	3	3	0	0	29	15.5	yes	8.93
6	rpmfind(by main.)	20	3	0	3	0	39	19	yes	15
7	tennis	10	-	-	-	-	-	-	no?	-
8	uefa(teams)	20	8	7	1	0	2	7.05	yes	11.8
9	uefa(play)	20	1	1	0	0	2	10.67	yes	22.1
10	wine(acc)	10	3	3	0	0	16	13	yes	68.2
11	wine(prod)	10	4	0	0	0	37	19.6	yes	120

There're still several examples that the system cannot build a perfect wrapper. On `Yahoo Shopping` in Table 1, there are two data items having different roles taking the same occurrence path (the only difference between the HTML path of them are the predicates on the text node, e.g. the difference between "//text[0]" and "//text[1]"), the system happened to identify them as a case of iteration. On `eBay(auction)`, the selected sample pages for inducing wrapper contain only 5 data items of total 6 items, resulting the wrapper cannot extract the left one from several pages. The induced wrapper for another example `uefa(teams)` in Table 2 can only partially extract one of the data items since the data item takes different occurrence paths ("//#text" vs. "//span/#text") in the two sample page, making the system falsely identifying a case of disjunction. And `rpmfind(by main.)` extract only partial information for all the 3 data items because the system can not discover all the semantic blocks.

7 Conclusion

In this paper, we propose a novel technique to the problem of item identification, which is one of the key problems in our automatic extraction approach. The identification problem is distinguished into simple identification and complex identification based on the complexity of the pages we face. The problem is resolved at various levels: semantic blocks, sections, path-groups and data items, and several approaches are proposed to effectively identify the mapping between them. Experiments on real pages have proved the effectiveness of the approach of item identification.

Future work will focus on the following two aspects: (i) How to automatically annotate the extracted data items; (ii) How to automatically detect fails in the process of extraction, and how to adopt user interactions for satisfactory results.

8   Acknowledgements

References

 1. Baumgartner, R., Flesca, S., Gottlob, G.: Visual web information extraction with lixto. In: Proceedings of VLDB. (2001) 119–128
 2. Liu, L., Pu, C., Han, W.: Xwrap: An xml-enabled wrapper construction system for web information sources. In: Proceedings of ICDE. (2000) 611–621
 3. Meng, X., Wang, H., Hu, D., Li, C.: A supervised visual wrapper generator for web-data extraction. In: Proceedings of COMPSAC. (2003) 657–662
 4. Sahuguet, A., Azavant, F.: Building intelligent web applications using lightweight wrappers. Data Knowl. Eng. **36** (2001) 283–316
 5. Muslea, I., Minton, S., Knoblock, C.A.: Hierarchical wrapper induction for semistructured information sources. Autonomous Agents and Multi-Agent Systems **4(1/2)** (2001) 93–114
 6. Arasu, A., Garcia-Molina, H.: Extracting structure data from web pages. In: Proceedings of SIGMOD. (2003) 337–348
 7. Crescenzi, V., Mecca, G., Merialdo, P.: Roadrunner: Towards automatic data extraction from large web sites. In: Proceedings of VLDB. (2001) 109–118
 8. Wang, J., Lochovsky, F.H.: Data extraction and label assignment for web databases. In: Proceedings of WWW. (2003) 187–196
 9. Grumbach, S., Mecca, G.: In search of the lost schema. In: Proceedings of ICDT. (1999) 314–331
10. : Xml query language (xquery). (http://www.w3.org/TR/xquery/)
11. : Xml path language (xpath) 2.0. (http://www.w3.org/TR/xpath20/)
12. : Document object model (dom) level 2 core specification. (http://www.w3.org/TR/DOM-Level-2-Core)
13. Arlotta, L., Crescenzi, V., Mecca, G., Merialdo, P.: Automatic annotation of data extracted from large web sites. In: Proceedings of WebDB. (2003) 7–12
14. Meng, X., Hu, D., Li, C.: Schema-guided wrapper maintenance for web-data extraction. In: Proceedings of ACM WIDM. (2003) 1–8
15. Meng, X., Wang, H., Hu, D., Gu, M.: Sg-wram: Schema guided wrapper maintenance. In: Proceedings of ICDE. (2003) 750–752
16. Lerman, K., Minton, S.: Learning the common structure of data. In: Proceedings of AAAI/IAAI. (2000) 609–614
17. Laender, A.H.F., Ribeiro-Neto, B.A., da Silva, A.S., Teixeira, J.S.: A brief survey of web data extraction tools. SIGMOD Record **31** (2002) 84–93