A Two-Phase Rule Generation and Optimization Approach for Wrapper Generation

Yanan Hao **Yanchun Zhang**

School of Computer Science and Mathematics
Victoria University
Melbourne, VIC, Australia

haoyn@csm.vu.edu.au

yzhang@csm.vu.edu.au

Abstract

Web information extraction is a fundamental issue for web information management and integrations. A common approach is to use wrappers to extract data from web pages or documents. However, a critical issue for wrapper development is how to generate extraction rules. In this paper, we propose a novel two-phase rule generation and optimization (2P-RULE) approach for wrapper generation. 2P-RULE consists of internal rule optimization (IRO) process and external rule optimization (ERO) process. In IRO, a user, through a GUI interface, firstly creates a mapping from useful values in web page to a schema specified by the users according to target web information. Based on the mapping, the system automatically generates a rule list for the schema. Whereas in ERO, the user can create multiple mappings to generate further rule lists. All the acquired rule lists are merged and refined into one optimized rule list, which is expressed with XQuery as the final extraction rules. Experiments show that our 2P-RULE approach is suitable for extracting information from web pages with complex nested structure, and can also achieve better precision and recall ratio[.]

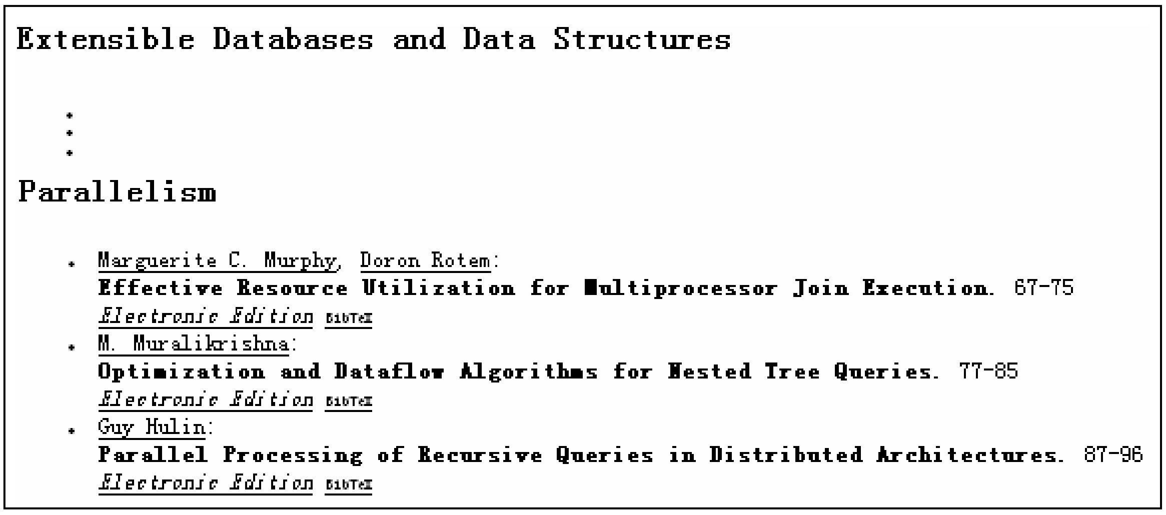
Keywords: Web, extraction, wrapper, rule optimization, XQuery.

1 Introduction

With the rapid development of Internet, World Wide Web has already become the most important and potential information resources (Lawrence S. and Giles L. 1999). HTML language aims at the visual presentation of data in web browsers, while it lacks of schema and semantic information for efficient management and retrieval web information. Most of valuable web information is in HTML form even though XML has been more and more popular today. So researchers propose wrappers technology to extract data from web pages and convert the information into a structured format. However, a critical issue for wrapper development is how to generate extraction rules for extracting data from web pages having similar structures.

Many information extraction tools (Alberto H. F. Laender, Berthier A. Ribeiro-Neto, Altigran S. da Silva, Juliana S. Teixeira., 2002) have been developed to extract data on the web. These works can be classified into three categories: manual approach, automatic approach (Soderland S. 1999, Arasu, A., Garcia-Molina, H. 2003, Hu D., Meng X. 2005, Ma L., Shepherd J. 2004) and semi-automatic approach (Liu L., Pu C., Han W. 2000, Han W., Buttler D., Pu C. 2001, Arnaud S. and Fabien A. 1999, Sahuguet A. and Azavant F. 1999, Baumgartner R., Flesca S., Gottlob G. 2001, Baumgartner R., Ceresna M., Gottlob G., Herzog M., Zigo V. 2003, Meng X., Wang H., Hu D., Chen L. 2003). In the first category, extraction rules are programmed manually which can be very hard for common users; in the second category, (Soderland S. 1999) introduces a machine learning approach. It utilizes the structures of sentences and relationships between idioms and words to create rules automatically; (Arasu A., Garcia-Molina, H. 2003) and (Hu D., Meng X. 2005) propose item identification techniques via HTML path and templates for automatic data extraction from web pages; (Ma L., Shepherd J. 2004) discovers the semantic pattern for an identified region of a document via inference, apposition and analogy. The drawbacks of these four systems are their limited expressive power of extraction rules and only suitable for simple record schema. Semi-automatic approach requires user interactions to build mappings between schema and content in web pages, after that extraction rules are derived for extracting web pages having similar structures. In the third category, XWrap (Liu L., Pu C., Han W. 2000, Han W., Buttler D., Pu C. 2001) only have good performance on web pages with distinct region features; In W4F (Arnaud S. and Fabien A. 1999, Sahuguet A. and Azavant F. 1999), expertise is required to program part of extraction rules manually; In Lixto (Baumgartner R., Flesca S., Gottlob G. 2001, Baumgartner R., Ceresna M., Gottlob G., Herzog M., Zigo V. 2003), the extraction rules are expressed in Elog language, which is difficult for the optimization and refinement of extraction rules; secondly, it require users to specify some external and internal conditions for extraction rules, thus the effectiveness and robustness of rules relies on user's action. SG-Wrapper (Meng X., Wang H., Hu D., Chen L. 2003) makes some improvements in rule generation and expression. But its extraction rules may be invalid when the nested structures of web page do not match the pre-defined user schema.

All the tools above ignore the problem of usable features of web page and their performance in constructing extraction rules. It is very important to the robustness of extraction rules. In this paper, based on the analysis of usable features of web pages and their performance in constructing extraction rules, we propose



Extensible Databases and Data Structures

•
•
•

Parallelism

- Marguerite C. Murphy, Doron Rotem:
  Effective Resource Utilization for Multiprocessor Join Execution. 67-75
  Electronic Edition BibTeX
- M. Muralikrishna:
  Optimization and Dataflow Algorithms for Nested Tree Queries. 77-85
  Electronic Edition BibTeX
- Guy Hulin:
  Parallel Processing of Recursive Queries in Distributed Architectures. 87-96
  Electronic Edition BibTeX

Figure 1: A sample HTML fragment from VLDB

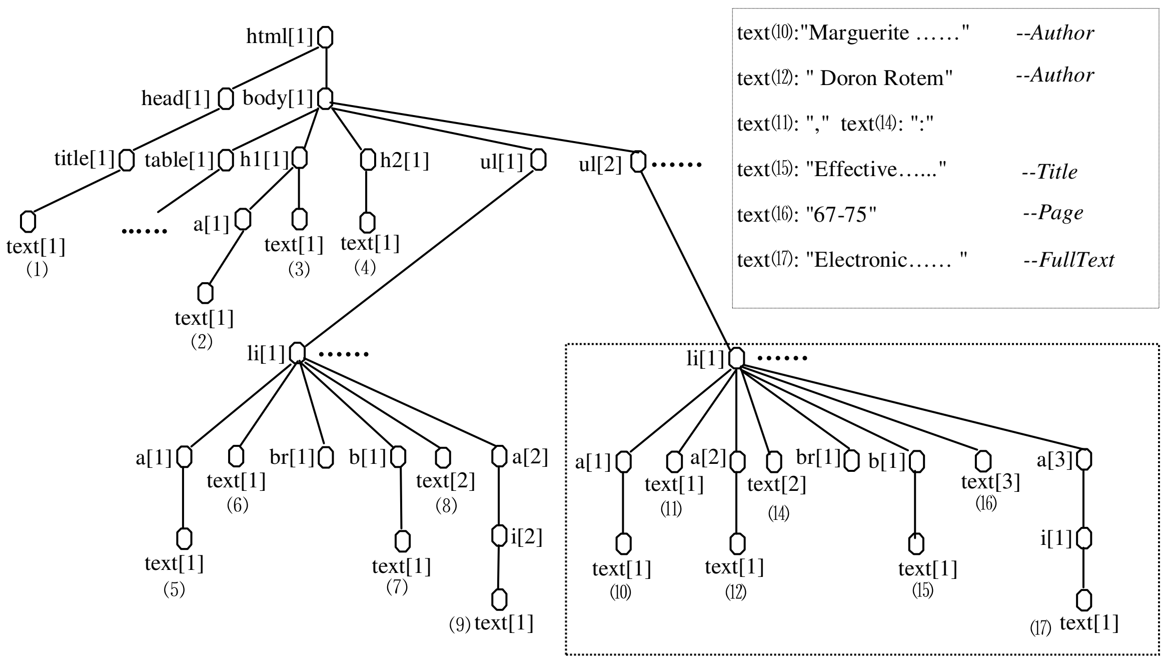
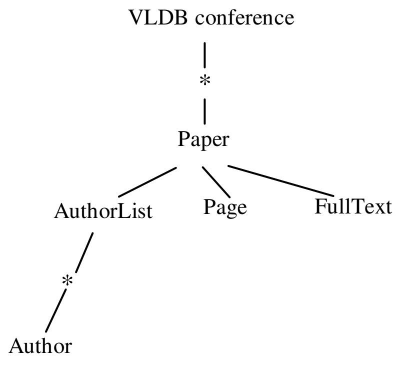


Figure 2: DOM tree



```
<!DOCTYPE  VLDBConference [
<!ELEMENT  VLDBConference  ((Paper)*)>
<!ELEMENT  Paper  (AuthorList, Page, FullText)>
<!ELEMENT  AuthorList ((Author)*) >
<!ELEMENT  Author  (#PCDATA)>
<!ELEMENT  Page  (#PCDATA)>
<!ELEMENT  FullText  (#PCDATA)> ]>
```

Figure 3: The schema tree and DTD definition

a novel two-phase rule generation and optimization (2P-RULE) approach. 2P-RULE consists of internal rule optimization (IRO) process and external rule optimization (ERO) process. In IRO, a user, through a GUI interface, firstly creates a mapping from useful values in web page to a schema specified by the users according to target web information. Based on the mapping, the system automatically generates a rule list for the schema. Whereas in ERO, the user can create multiple mappings to generate further rule lists. All the acquired rule lists are merged and refined into one optimized rule list, which is expressed with XQuery as the final extraction rules. Then the information extraction is simply a process of executing XQuery statements in any XQuery engine. The query result can be utilized by common users and further by applications. Experiments show that our 2P-RULE approach can extract information from web pages with complex nested structure and can also achieve better precision and recall ratio.

This rest of this paper is organized as follows. Section 2 introduces the presentation and semantic models for web information extraction. In section 3 we present the basis of extraction rules generation. Section 4 discusses the extraction rules and their optimization steps. Section 5 reports the experimental results. Finally conclusion and future work are discussed in section 6.

2 Representation and semantic models

2.1 Representation model of web documents

An HTML document is a text file containing markup tags. The Document Object Model (DOM) represents a document as a tree. Every node of the tree represents a HTML tag, or a text value inside an HTML tag. The tree structure describes the whole HTML document, including the child, parent, or sibling relationship between tags and text values on the page. DOM allows us to locate elements in the tree with XPath (XML Path Language) expressions. We choose DOM as the representation model of HTML information in our system. All the operations in our system are based on DOM tree.

As an example, in Figure 1 we give an HTML fragment of the web pages at http://www.informatik.uni-trier.de/~ley/db/conf/vldb/. Figure 2 shows its DOM tree structure. In Figure 2 each node tag is followed by an *ordinal*. An ordinal is the order of a node among all its siblings of the same tag. For the convenience of reference, we also assign each text node a number. For example, the following XPath expression "html[1]/body[1]/ul[2]/li[1]/text()[3]" identifies the data value 67-75 in the HTML fragment.

2.2 Semantic model

DOM tree is only the internal expression of web documents. It can effectively process data in documents, but it may not reflect the potential semantic information in document data. In this paper we choose XML as the semantic model and its schema is defined by DTD. The defined DTD can be easily represented in the form of a tree structure, which is called a **schema tree.** In our system, through a GUI interface, a user can easily specify the schema tree according to the target web documents. An example of the schema tree and the DTD corresponding to that can be seen in Figure 3. The '*' in Figure 3 denotes zero or more occurrences. For example, the '*' between *AuthorList* and *Author* means a *AuthorList* may have zero or more authors. The schema tree can support some node types corresponding to the data types defined by DTD. Using regular expression, we define these supported node types as follows:

atomic object (AO): (#PCDATA)
set object (SO): ((atomic object)*)| ((tuple object)*)
tuple object (TO): (a_1, a_2… a_n), where a_i (1<=i<=n) is (an atomic object | a set object | a tuple object)

We call each sub element of a set object a **member object (MO).** If the member is an atomic object, then it will be called a **member atomic object (MAO***)*. If the member is a tuple, then it is a **member tuple object (MTO).** In general, all nodes in the schema tree are called **semantic objects.**

For example, referring the schema tree in Figure 3, both *Page* and *FullText* are atomic objects (AO); *VLDBconference* and *AuthorList* are set objects (SO); A*uthor* is a member atomic object (MAO); and *Paper* is a member tuple object (MTO).

After creating a schema tree, the user needs to create a mapping from the contents of web pages to the schema tree. To create such a mapping, the user simply selects semantic objects in the schema tree, and then highlights the corresponding content on web pages. Based on the mapping, the system automatically generates *rule segments* (see section 4) for each semantic object of the schema tree.

3 Basis of extraction rules generation

3.1 Analysis of usable web features

In most information extraction systems, extraction rules are mainly expressed with five features of web pages, including structure, position, semantics, display and references. Feature selection determines the performance of extraction rules. *Structure feature* is the paths of DOM tree. With path expression we can navigate HTML page easily, so it can be a basic feature for constructing extraction rules. But structure feature has weak differentiating ability, and extraction rules only containing structure feature may have low precision rate. For example in Figure 2, the path expression html/body/ul/li/text() can locate many nodes. *Position feature* includes *ordinal* (section 2.1) and *boundary*. Boundary is a left or right sibling node. In Figure 2, node *text(16)* has the left boundary *b* and the right boundary *a*. *Position feature* relies on the structure of web pages, so using position will decrease the coverage of extraction rules but increase the differentiating ability at the same time. *Display feature* includes font, font-size, colour and alignment. It limits nodes by node attributes in each location step of DOM path. Normally, similar or correlative contents in web pages have same display features, so selecting display feature generating rules will increase the coverage. But extraction rules can be inaccurate when multiple instances of one semantic object (say an *author* object) have same display features. *Semantic feature* is a common conceptual or content feature of data to be extracted. For example, the value of *Price* often contains a character '$'. Semantic feature lacks in differentiating ability but makes extraction rules more flexible. *Reference feature* is the hyperlink information in web pages. It has little effect on the robustness of extraction rules and we do not discuss it in this paper.

The goal of extraction rules is to have good coverage and differentiating ability. Based the analysis above, we firstly select DOM path, semantic and display features to form extraction rules, which do not rely on the structure of web pages and have good coverage ability, then add ordinal and position information to extraction rules to gain good differentiating ability.

3.2 Mismatches between schema tree and DOM tree

To extract data from similar web pages or documents, a user first defines a schema tree of the target data. The schema tree reflects the user's view of extracted data. In order to make semantics clear, the user can create a schema tree with nested structure. For example in Figure 3, a *AuthorList* object is created to represent all the authors in one paper. Since the user creates a schema tree through a GUI interface without knowing the details of HTML documents, sometimes the nested structure of the schema tree may not match its corresponding structure in DOM tree. The mismatch happens when a complex nested object in the schema tree does not have a corresponding node in DOM tree, while its sub components are directly listed. For example in Figure 3, *AuthorList* is a complex nested object whose sub components are *Author*s. Consider the DOM tree in Figure 2. The sub-tree in rectangle corresponds to the semantic object *Paper*, and its DOM paths correspond to *author*, *author*, *title*, *page* and *FullText* respectively. No node corresponds to the semantic object *AuthorList* and all the *Author* nodes are listed directly.

Mismatches between schema tree and DOM tree are main difficulties for extracting complex nested objects. Typically, there are three sorts of mismatches:

Single set object mismatch. There is only one set object in the schema tree that does not match its corresponding DOM tree structure, for example the *AuthorList* object in Figure 3. Let us suppose *author* and *FullText* have same display and structure features firstly. Further more, since the count of authors is variable, position feature can not differentiate them either. In this case, we introduce a new position feature *big boundary*. The *left big boundary* of a set object is a sequence of nodes, which are all the left siblings of the leftmost sub-tree spanned by the set object; the *right big boundary* of a set object is a sequence of nodes, which are all the right siblings of the rightmost sub-tree spanned by the set object. For example in Figure 2, the set object *AuthorList* corresponds to nodes text (10) and text (12). The *left big boundary* of *AuthorList* is null while its right *big boundary* is {text, br, b, text, a}. By *big boundary* feature, we can differentiate between member objects (*Author*) of a set object (*AuthorList*) and sibling objects (*FullText* or Page) of the set object.

Multiple set objects mismatch. There are several set objects in the schema tree, and none of them matches the corresponding DOM tree structure. For example, a user may define another set object *AddressList* as a sibling of *AuthorList*. Using the big boundary, we can still differentiate these two set objects if member object *Author* and *Address* can be differentiated. But if *Author* and *Address* have same features, these two set objects can not be differentiated in our system.

Member tuple object mismatch. In this case, a member tuple object does not have the corresponding node in DOM tree while its sub components are listed directly. The member tuple objects may not be differentiated. For example in Figure 2, if the node *Li[1]* , which corresponds to the semantic object *Paper*, does not exist, all the authors can not be differentiated, i.e. which author belongs to which paper. In our system, we add a virtual node to each member tuple object to solve the *member tuple object mismatch.*

In this section, we analyse the usable web features and the mismatches between schema tree and DOM tree. They are the basis of rules generation in our system. In section 4, we will describe our approach to generate and optimize extraction rules.

4 Generation and optimization

4.1 Rule segments

According to section 3, we distribute all the usable web features in six sorts of rule segments. The initial extraction rule for each semantic object will be composed of several rule segments. Different semantic object has different composition of rule segments as the initial extraction rule. Figure 4 gives the BNF definition of all rule segments.

PureAttrPathExp(P): We use the first letter **P** to denote PureAttrPathExp. Abbreviation for other rule segments is similar. This rule segment is called *pure-attribute path expression*, each location step of which only contains attributes limitation. If there exists attributes in a location step, then we choose all the equations of "[attribute name= *attribute value*]" as predicates to limit nodes sequence, or we do not select any predicates in this step. For example in Figure 2, html/body[@bgcolor="#ffffff"]/ul/li/text() is a pure-attribute path expression. It can locate text(11), text(16) and text(14).

AttrOrdPathExp(A): We call it *attribute-ordinal path expression*, each location step of which only contains attributes or ordinal limitation(except location steps with the node test *text()*). If it contains attributes, then we use all the equations "[attribute name= *attribute value*]" as predicates, or we use ordinals to limit nodes sequence. In Figure 2, html[1]/body[@bgcolor="#ffffff"]/ul[2]/li[1]/text() is an *attribute-ordinal path expression.*

OrdPathExp(O): This sort of rule segment is called *ordinal-path expression,* each location step of which only contains ordinal limitation (including location steps with the node test *text()*). For example,

```
PureAttrPathExp::= (NodeName("[@"AttrName"="AttrValue"]")* ) |
(NodeName("[@"AttrName"="AttrValue"]")*"/"PureAttrPathExp) | NULL
AttrOrdPathExp::=(NodeName("["num"]" | ("[@"AttrName"="AttrValue"]")*)) | NULL
| (NodeName("["num"]" | ("[@"AttrName"="AttrValue"]")*)"/"AttrOrdPathExp)
OrdPathExp::=(NodeName"["num"]") | (NodeName"["num"]""/"OrdPathExp) | NULL
TxtFeaturePredicate::=("[contains(string(.) ," TxtFeatureValue ")]") +
Big_BoundaryPredicate::=(("[count(../" NodeName "after .)=" Num "]" ) | ("[count(../" NodeName "before .)="
Num "]")) +
Small_BoundaryPredicate::=
(("[count((../* after .)[1])=0 ]") | ("[((../"Nodename " after" ".)[1])=((../*  after .)[1])]") )
(("[count((../*  before .)[1])=0 ]") | ("[((../"Nodename " before" ".)[1])=((../*  before .)[1])]"))
```

Figure 4: The BNF definition of rule segments

html[1]/body[1]/ul[2]/li[1]/text()[3] is a OrdPathExp for the semantic object *Page*(see Figure 2) .

OrdPathExp(O): This sort of rule segment is called *ordinal-path expression,* each location step of which only contains ordinal limitation (including location steps with the node test *text()*).For example, html[1]/body[1]/ul[2] /li[1]/text()[3] is a OrdPathExp for the semantic object *Page*(see Figure 2) .

TextFeaturePredicate(T): This rule segment is called *text-feature predicate.* It is a form of predicate in XPath requiring the text content of a node (for non-leaf node, it will be a concatenation of string-values of all its descendants) in DOM tree contain some fixed text value. We only use this predicate in the last location step of path expressions. For example, html/body[@bgcolor=”#ffffff”] /ul/li[contains(string(.)，”Electronic Edition”)] means the text content of nodes limited by the last location step must contain the string “Electronic Edition”.

Big _BoundaryPredicate(B): This rule segment is called *big boundary predicate.* It contains *left big boundary predicate* and *right big boundary predicate*. We introduce this rule segment for the extraction of complex nested objects, say *AuthorList* in Figure 3(section 3.2).

Small_BoundaryPredicate(S): This rule segment is called *small boundary predicate*. This predicate is to limit a node by its immediate left sibling and right sibling. In our system we only apply this segment to atomic objects and only to the last location step with the node test “text ()”, because text nodes in DOM specification are regarded as virtual nodes, and in most circumstances they do not have sibling nodes, as they do not rely on the structure of web pages.

In these six rule segments, the first three rule segments are path expressions, in which PureAttrPathExp has the best coverage ability, OrdPathExp has the worst coverage ability, and AttrOrdPathExp is middle; the rest three rule segments are all predicates. They can only be used together with the first three path expressions.

After the user creates a mapping from the contents of web pages to the schema tree, system automatically generates rule segments for each semantic object of the schema tree. As for a member object, the two segments *AttrOrdPathExp* and *OrdPathExp* do not contribute to its extraction rule, since they both contain ordinal feature and it can be invalid due to the variable count of member objects. System does not generate rule segments for a plain tuple object (i.e. it is not a member tuple object). The plain tuple object appears only once, and if its sub components can be extracted, they definitely belong to this tuple object. Thus the extraction rule for plain tuple object itself is not needed and we only need to compose rules for its sub components. But for a member tuple object, it can appear many times, so the corresponding extraction rule is needed to decide which sub component belongs to which member tuple object instance.

The rule segments of each semantic object are marked by “ √ ”in Table 1. Please see section 2.2 for the definition of semantic objects. For example, from Figure 2 we can conclude that html/body[@bgcolor=“#ffffff”]/ul/li/text() is a rule segment (P) for the semantic object *Author* (MAO).

Semantic objects / **Rule segments**	**AO**	**MAO**	**MTO**	**SO**
PureAttrPathExp	√	√	√	√
AttrOrdPathExp	√			√
OrdPathExp	√			√
TxtFeaturePredicate	√	√	√	√
Big_ BoundaryPredicate				√
Small_BoundaryPredicate	√	√		

Table 1: Rule segments for each semantic object

4.2 Optimization of extraction rules

For one semantic object, its rule segments can have different combinations. Each combination is called an *initial rule* for the semantic object. Different combination of rule segments can generate different initial extraction rules. In this section, we describe our two-phase rules

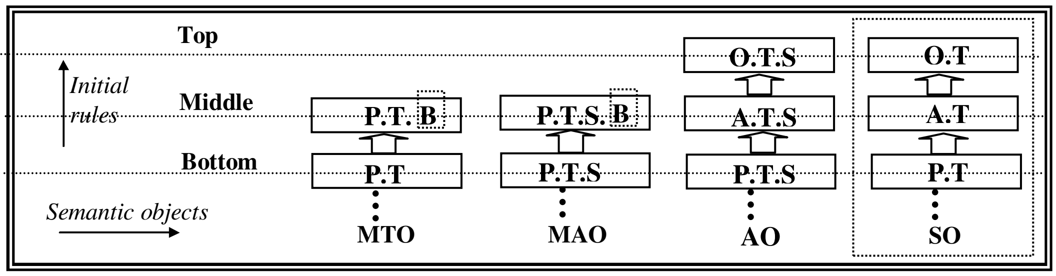


Figure 5: Initial rules for each semantic object

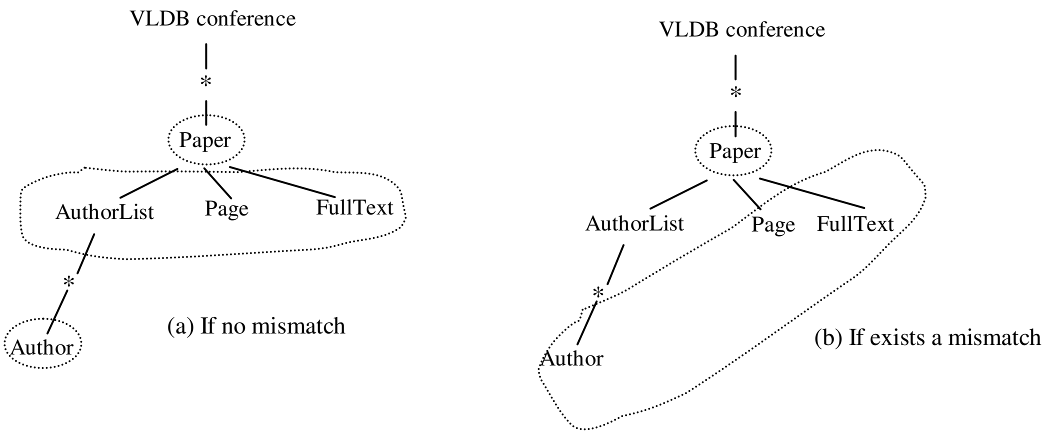


Figure 6: Grouping of semantic objects

(2P-RULE) optimization approach. 2P-RULE consists of the internal rule optimization (IRO) process and the external rule optimization (ERO) process.

4.2.1 Internal optimization (IRO)

In IRO, system automatically selects an optimized initial rule for each semantic object and generates a rule list for the schema.

(1) Initial rules

Among the six rule segments, the first three path expressions are XPath statements, and the rest three rule segments are predicates. The effective combination, initial rule, should be one and only one path expression plus several predicates. We combine both TxtFeaturePredicate and Small_BoundaryPredicate together with path expressions, for these two segments do not rely on the structure of web page. For Big_BoundaryPredicate, it is generated to deal with *set object mismatch*. When there exist set objects mismatches, we do not generate initial rules for SO and add its Big_BoundaryPredicate to member objects MTO and MAO. While when there are no mismatches, the initial rules for SO are {P.T, A.T, O.T} and the Big_BoundaryPredicate is removed. All the possible initial rules for each semantic object are listed in Figure 5.

Bottom. The initial rules at bottom have better coverage ability. They select DOM path, display feature and semantic feature. Although S rule segment is added to the atomic object, it does not rely much on the structure of web page. The shortcoming of bottom combination is that it lacks of differentiating ability.

Middle. The middle level of initial rules are acquired by adding position feature to bottom combination to get better differentiating ability. Particularly, member objects get new rule segment B; both atomic objects and set objects replace their rule segment P as A, which actually complements some ordinals for the location steps without predicates. The better differentiating ability is at the expense of decreasing of coverage.

Top. The top initial rules are only available for atomic objects and set objects. They add further position feature to the middle initial rules. Comparing with middle initial rules, their differentiating ability is further increased but coverage ability is further decreased.

(2) Optimization and selection

The goal of extraction rules is to have good coverage and differentiating ability. In Figure 5 we can see, from bottom to top, the coverage ability of initial rules is increasingly good and the differentiating ability of initial rules is increasingly poor. Our basic idea of rules optimization is to select the first "good" initial rule from bottom to top for each semantic object. Here "good" means having no collision with the selected initial rules for other semantic objects, i.e. they do not locate identical nodes.

In our system, the DOM paths forming into extraction rules are relative. Extraction rules for sub objects will locate nodes based on the extraction results of parent objects, so only the initial rules having the same base domain are possible to collide. For example in Figure 2, suppose "html/body/ul/li" is an initial rule for *Paper*, and the initial rule for *FullText* is "a/i/text()". Obviously, these two initial rules do not collide, because "a/i/text()" is to locate nodes based on the sub tree *li*, and "html/body/ul/li" is to locate nodes based on the whole DOM tree. To detect potential collisions, we group the semantic objects according to the base domain of their initial rules. The semantic objects having sibling relationship will belong to the same group finally.

For example, Figure 6 gives the two possible groupings of schema tree in Figure 3. For the HTML fragment in Figure 1, we should choose the grouping (b).

Definition 1 [Containment relationship] Let XPathExp1 and XPathExp2 be two XPath expressions. We say XPathExp1 contains XPathExp2, if

1. They have the same DOM path, and
2. The set of predicates in each location step of XPathExp1 is a subset of the set of predicates in each corresponding location step of XPathExp2.

The containment relationship is denoted as XPathExp2 $\subset$ XPathExp1. For example, suppose XPathExp1=A/B[@colour="1"]/C, and XPathExp2=A/B [@calor="1"][@high="6"]/C, then XPathExp2 $\subset$ XPathExp1. This means the nodes set located by XPathExp2 are a subset of nodes set located by XPathExp1 and so collision occurs. Semantic objects in same group have the same DOM path, so we can use containment relationship to detect potential collisions between them.

Definition 2 [Invalid rule] Let so_i be the *i*th semantic object in a schema tree, $max(so_i)$ be the total count of initial rules for so_i and so_i-$r_m(1 \leqslant m \leqslant MAX(SO_i))$ be the *i*th initial rule for so_i. We say so_i-r_m is an invalid rule, if and only if $\exists$ so_j ($j \neq i$) such that so_i-r_m contains so_j-$r_n(1=<n<=max(so_j))$. Here *m* and *n* are the ordinals of initial rules for semantic objects. For each semantic object, its initial rules are numbered from bottom to top beginning with 0 (See Figure 5).

We say that an initial rule is valid if it is not an invalid rule. Based on the analysis and definitions above, the process of selection and optimization of rules for semantic objects is described as below:

Step 1: Group all the semantic objects.
Step 2: Group all the initial rules by the grouping of semantic objects.
Step 3: In each initial rules group, from bottom to top, Find the first valid initial rule as the optimized rule for each semantic object. All the optimized rules constitute an optimized rule list for the schema. We formulate this step into algorithm 1.

Algorithm 1 Internal optimization

Input: G= {g_i | i=1, 2 …}, groups obtained in step 2
Output: O= {opti | i=1, 2 …}, O is an optimized rule list for the schema, in which opt_i contains the optimized rules for semantic objects in group g_i

Description:

```
for each group g_i in G do
  Optimize (g_i);
Function optimize (g)
1: s:=0; collision:=False; // s is the ordinal of an initial rule
2: for each semantic object $so_l$ in group g do
3:   if max ($so_l$) =1 then
4:     // only one initial rule
5:       $opt_g$-$so_l$ = $so_l$ –$r_0$ ;
6:       continue;
7:   endif
8:   if exists i, j such that $so_j$-$r_i \subset so_l$-$r_s$ then
9:     // $so_l$-$r_s$ is an invalid rule
10:    collision: =True;
11:  endif
12:  if collision then
13:      if ($so_l$ is a MO) and s=1 then
14:          exit; // Can not extract this web page.
15:      else
16:          s: =s+1; // go up
17:          goto line 8;
18:      endif // if $so_l$ is a member object
19:  endif // if collision
20:  $opt_g$-$so_l$ = $so_l$-$r_s$; // Obtain the optimized rule for $so_l$
21:  $opt_g$-$so_l \rightarrow opt_g$; //add $opt_g$-$so_l$ to $opt_g$
22: end // end for
```

For example, after IRO we can obtain an optimized rule list for the semantic objects in Figure 3:

- *Paper*: html/body[@bgcolor="#ffffff"]/ul/li [contains(string(.), "Electronic Edition")]
- *Author*: a[left_big_boundary="", right_big_boundary ="text, br, b, text, a"]/text()
- *Page*: /text()
- *FullText*: a/li/text()[contains(string(.), "Electronic Edition")]

4.2.2 External optimization (ERO)

We say a user makes a learning process, if the user creates a mapping from the web page to the schema. Based on the mapping, system generates an optimized rule list by IRO. If the user is not satisfied with the extraction results, she can make more sample learning processes. Each sample learning process will generate one optimized rule list. In ERO system merges and refines all the acquired rule lists into one optimized rule list, which is expressed with XQuery as the final extraction rules.

The whole merging and refining procedure is listed as below:

Step 1: For each semantic object.
(1) Find all its relevant optimized rules
(2) Partition these rules into several groups. In each group the rules have containment relationship with each other.
(3) Select the rule having the best coverage in each group and unit them into an optimized rule for the semantic object.

Step 2: Save all the optimized rules as the final optimized rule list for the schema.

We formulate these steps into algorithm 2.

Algorithm 2 External optimization

```
Input: Optimized rule lists: $opt_1, opt_2, ..., opt_k$
       $Opt_i = \{opt_i\text{-}so_1, opt_i\text{-}so_2, \dots, opt_i\text{-}so_n\}$ $(0<i<k+1)$
Output: the final optimized rule list Opt for the schema.
Opt= $\{opt\text{-}so_1, opt\text{-}so_2, \dots, opt\text{-}so_n\}$
Description:
1: for $m$=1 to n do
2:   partition the set $\{opt_l\text{-}so_m | 1<=l<=k\}$ into $s$ groups.
     In the $i$th $(1<=i<=s)$ group, find $opt_{li}\text{-}so_m$ that contains
     all the other extraction rules for the semantic object $so_m$
3:   $opt\text{-}so_m = \bigcup_{i=1}^{s} (opt_{li} - so_m)$
4: endfor
```

4.3 XQuery expression

After ERO, we get the final optimized rule list for the schema. Each rule of the optimized rule list is expressed with an XPath expression, and each time it can only locate in DOM tree one semantic object instance of the schema. In order to locate all the semantic object instances, we translate the final rule list into a complete XQuery query statement as the extraction rule for the schema.

According to the final optimized rule list, we generate one FLWR expression for each semantic object, i.e.

- One FR expression (FOR statement and RETURN statement) for a member object.
- One LR expression (LET statement and RETURN statement) for a set object and an atomic object.

Finally, we organize all the FLWR expressions by the nested structure of the schema tree and form them into one XQuery statement. The information extraction is then a procedure of executing this XQuery statement in any XQuery engine.

As an example, let us suppose the final optimized rule list is the same as it in section 4.2.1. The XQuery statement is shown in Figure 7.

```
<vldb conference>
{FOR $paper IN (document("sample.xml")/html/body
 [@bgcolor="#ffffff"]/ul/li[contains(string(.),"Electronic
Edition")])
 RETURN
 <Paper>
  <AuthorList>
  {FOR $author IN $paper/ a[left_big_boundary="",
   right_big_boundary ="text, br, b, text, a"]/text()
   RETURN
    <Author>{$author}</Author>}
   </AuthorList>
  {LET $page:=$paper/text()
   RETURN
  <Page>{$page}</Page>}
  {LET $FullText:= $paper/ a/li/text()
   [contains(string(.),"Electronic Edition")]
   RETURN
  <FullText>{$FullText}</FullText>}
 </Paper>}
</ vldb conference>
```

Figure 7: Extraction rules expressed with XQuery

5 Experiments

Based on the optimization techniques above, we have developed a prototype system. Several experiments have been done on the three websites sites in table 2, which are already used for testing purposes by other information extraction tools.

We carry out our experiments on a Windows machine with a 2GHz Pentium IV and 512M main memory. For each website, the experiment procedure is listed as below:

Transformation. Because our prototype system uses XML as the presentation model of HTML information in web pages, and all the operations are based on DOM tree, all the web pages to be extracted to XML documents should be transformed into XML documents firstly. We use Tidy (HTML Tidy Library Project) to finish the transformation.

Creating schema tree. Select sample web pages, then require the user to create schema tree to represent the semantic information of data to be extracted.

Creating mappings. The user selects semantic objects in the schema tree firstly, and then highlights the corresponding content on web pages. Meanwhile text feature may be required.

Optimization. Execute the IRO and ERO process. System automatically generates optimized rule list expressed with XQuery.

Extraction. Execute the XQuery statement on an XQuery engine to extract data on other web pages in the website.

Analysis. We manually verify the extracted results.

5.1 Evaluation metrics

We use recall and precision rate to evaluate the effectiveness of our optimization approach. The recall and precision are defined as

- precision = A/(A+B)*100%
- recall = A/(A+C)*100%

Where A stands for the number of relevant objects, B stands for the number of irrelevant objects, C stands for the number of missing objects, A+C stands for the total number of relevant objects, and A+B stands for the total number of extracted objects.

5.2 Results analysis

Table 3 shows our 2P-RULE extraction results on the pages of websites collected in Table 2. Table 4 shows the extraction results of typical system Lixto. For the third website in Table 2, there is one webpage having 284 complex objects. Experiment results show that system extracts 285 complex objects totally with only one irrelevant object, so the precision is 99% (284/285) and the recall is 100% (no missing objects). The precision does not reach 100%, because user does not provide definite semantic information during the first learning. One object is extracted regarded as invalid by user. For the website

Name	URL	Webpage	Number of pages
Amazon	http://www.amazon.com	Top Sellers(TVs)	12
VLDB	http://www.acm.org/sigmod/dblp/db/conf/vldb	VLDB 1989	20
Web Robot	http://www.robotstxt.org/wc/active/html	Web Robot	1

Table 2: Test websites

Name	Wrapable?	Learning times	Precision	Recall	Test pages
Amazon	Yes	1	100%	98.3%	12
		2	100%	100%	12
Web Robot	Yes	1	99%	100%	1
		2	100%	100%	1
VLDB	Yes	1	100%	100%	20

Table 3: Experiment results of 2P-RULE

Name	Wrapable?	Learning times	Precision	Recall	Test pages
Amazon	Yes	1	95%	90%	12
		2	98%	100%	12
Web Robot	Yes	1	90%	96%	1
		2	95%	98%	1
VLDB	Yes	1	80%	90%	20

Table 4: Experiment results of Lixto

VLDB, our system works well. Its web pages contain complex semantic schema structure and the set objects do not have corresponding nodes in DOM tree. But after learning once, our system still has 100% precision when extracting 20 pages. For the Amazon website, there is an average recall of 98.3% on 12 web pages. The missing objects are due to the design of web page. We find that the text feature selected by user does not appear in some pages containing relevant objects and these objects are missed when applying extraction rules into their host pages. After learning once again, system automatically adds new text feature into extraction rules. So the missed objects are back and the recall becomes 100%. On this website Lixto achieves precision of 95% after learning once, and 100% after learning three times. Obviously, our system has better performance in recall, precision and learning times.

6 Conclusion and future work

In this paper, we propose a novel two-phase rule generation and optimization (2P-RULE) approach. 2P-RULE consists of internal rule optimization (IRO) process and external rule optimization (ERO) process. In IRO, based on the mapping created by a user, system automatically generates an optimized rule list for the schema. Whereas in ERO, the user can create multiple mappings to generate further rule lists. All the acquired rule lists are merged and refined into one optimized rule list, which is expressed with XQuery as the final extraction rules. Experiments show that our 2P-RULE approach is suitable for extracting information from web pages with complex nested structure, and can also achieve better precision and recall ratio. Our future work includes the automatic verification of extraction rules, the efficient organization, storage and management of obtained extraction rules.

References

Lawrence S., Giles L.(1999):Accessibility and distribution of information on the Web. Nature, 1999, 400(8): 107-109.

Alberto H. F. Laender, Berthier A. Ribeiro-Neto, Altigran S. da Silva, Juliana S. Teixeira. (2002): A Brief Survey of Web Data Extraction Tools. SIGMOD Record, 2002, 31(2): 84 - 93.

Soderland S. (1999): Learning Information Extraction Rules for Semi-structured and Free Text. Machine Learning, 1999, 34(1-3):233-272.

Liu L., Pu C., Han W. (2000): XWRAP: An XML-enabled Wrapper Construction System for Web Information Sources. In Proc. of the 16th ICDE Conf., San Diego, California, USA, 2000.

Han W., Buttler D., Pu C. (2001): Wrapping Web Data into XML. SIGMOD Record, 2001, 30(3):33-39.

Arnaud S. and Fabien A. (1999): Building Light-Weight Wrappers for Legacy Web Data-Sources Using W4F.In Proceedings of 25th VLDB Conference, Edinburgh, Scotland, UK, 1999.

Sahuguet A. and Azavant F. (1999): Web Ecology: Recycling HTML Pages as XML Documents Using W4F. In Second International. Workshop on the Web and Databases, Philadelphia, Pennsylvania, USA, 1999.

Baumgartner R., Flesca S., Gottlob G. (2001): Visual Web Information Extraction with Lixto. Proceedings of 27th International Conference on Very Large Database. Roma, Italy, 2001.

Baumgartner R., Ceresna M., Gottlob G., Herzog M., Zigo V. (2003): Web Information Acquisition with Lixto Suite. In Proceedings of the 19th ICDE Conference, Bangalore, India, 2003.

Meng X., Wang H., Hu D., Chen L. (2003): A Supervised Visual Wrapper Generator for Web-Data Extraction. COMPSAC 2003: 657-662

Arasu A., Garcia-Molina, H. (2003): Extracting structure data from web pages. In: Proceedings of SIGMOD. (2003) 337–348.

Hu D., Meng X.(2005): Automatically extracting data from data-rich web pages. DASFAA 2005, Beijing, 2005,4

Ma L., Shepherd J. (2004): Information Extraction via Automatic Pattern Discovery in Identified Region. DEXA 2004: 232-242

DOM.http://www.w3c.org/TR/REC-DOM-Level-1.Xpath

XML Path Language Version 2.0. http://www.w3.org/TR/xpath20

XQuery. http://www.w3.org/TR/xquery

HTML Tidy Library Project. http://tidy.sourceforge.net/