WebKER: Towards Information Extraction Base on Suffix Arrays and Domain Ontologies

Xi Bai[1,2], Jigui Sun[1,2]

[1] College of Computer Science and Technology
Jilin University
Changchun 130012, China
[2] Key Laboratory of Symbolic Computation and Knowledge Engineering
Jilin University
Changchun 130012, China
xibai@email.jlu.edu.cn

Abstract. Wrappers are widely used for extracting information from semi-structured Web documents. However, since resources are formatted in diverse ways for human viewing, sometimes the accuracy of extracting information by using wrappers are not satisfactory and users can not query the information easily. Based on pattern discovery, this paper proposes an ontology-based approach for automatically extracting information from Web pages in Chinese language. In the extraction process, wrappers are learned by using suffix arrays and then domain ontologies align the raw data extracted by wrappers from Web pages with distinct layouts. The aligned individuals are described with Resource Description Framework (RDF) statements and added in the knowledge base finally. So users can query them regardless of resources' derivation. In the experiments, the performance of our approach and the comparison between querying information extracted with the assistant of domain ontologies and querying information stored in traditional database are presented, indicating the superiority of our approach. In addition, the evaluation of the outstanding wrapper and the method for merging knowledge are also presented.

1 Introduction

With the emergence of the World Wide Web, making sense of large amount of information on the Web becomes increasingly important. Since most of the Web pages are written with the formatting language – HTML nowadays, when users do some queries, the search engine cannot understand the semantics of the searching results or make the data sensible. Recently, Information Extraction (IE) techniques are more and more widely used for extracting relevant data from Web documents. The goal of an IE system is to find and link the relevant information while ignoring the extraneous and irrelevant one [1]. Among these techniques, wrappers are widely used for converting Web pages to structured data. However, the layouts of Web pages from different Web sites are usually different. Moreover, the layout of a specific Web page may be updated from time

to time. Therefore, to find out a genetic method for automatically or semiautomatically generating wrappers becomes one of the hottest issues in information community. Several tools have already been applied in generating wrappers [6–8], however, since resources are typically formatted in diverse ways for human viewing, sometimes the accurate of extraction is not satisfactory and the traditional way of storing information is not convenient for users to query.

Recently, ontology-aided semantic annotation is used for achieving the annotation of pages with semantic information, which is the most important step in the IE procedure. In this paper, we expect to apply domain ontologies to assist wrappers in extracting information from Web documents. We propose a framework for extracting information from Chinese Web pages by using wrappers which are generated based on pattern discovery and domain ontologies. The remainder of this paper is organized as follows. Section 2 describes the related works of recent information extraction approaches. Section 3 describes the framework for our approach. Section 4 describes method of Web pages normalization. Section 5 describes the raw-information-extraction process and Section 6 describes the process knowledge generation and knowledge merging. Section 7 gives the experiments in which the performance for our approach and the comparison between querying information extracted with the assistant of domain ontologies and querying information stored in traditional database are presented. Finally, some concluding remarks and our future research directions are described in section 8.

2 Related Works

The goal of information extraction is to identify and extract specific resource fragments from corpus. The lack of homogeneity in the structure of the source data in the Web sites becomes a most obvious problem in designing a system for Web IE. With the growth of the amount of information on the Web, the availability of robust IE tools becomes necessary. Traditional IE concentrates on domains consisting of grammatical prose. Some researchers try to extract information using databases. It is known that the data in database is structured. However, most data on the Web is semi-structured or unstructured. So the conventional database cannot do the extraction work and a method is proposed to extract data from Web sources to populate databases for further handling [3]. Some IE systems have also been proposed, which are usually based on wrappers and take a single approach or attack a particular kind of domain. Wrappers are specialized programs which identify data of interests and map them to some suitable format [5]. At the beginning, wrapper generation is time consuming and tedious since it is usually done manually. Recently, some wrapper induction systems are proposed by researchers. TSIMMIS [6] generates wrappers using rules based on the users' specification. Some systems are based on heuristics wrapper induction such as STALKER [7] which is tailored for well-structured documents and WHISK [8] which is tailored for less rigorous structured documents. Ontology-based wrapper induction generates wrappers by defining and applying

the domain knowledge [9]. Generally speaking, most of recently developed wrappers are usually based on query languages [10], HTML structure analysis [11], grammar rules [12], natural language processing [13], machine learning [14], data modeling [15], ontologies [16], etc.

3 Framework

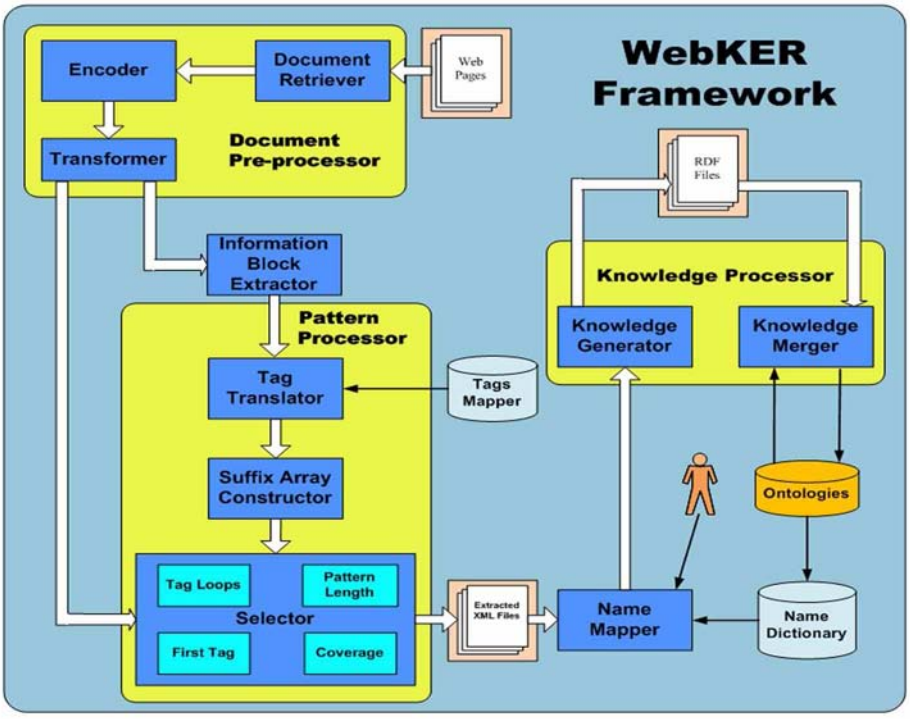

Fig. 1. Framework

In this section, we describe our framework for extracting information from Web documents in Chinese language. The framework is shown in Figure 1. In this figure, *Retriever* module retrieves Web documents from internet. Then these documents are transformed from ill-formed expression into well-formed expression by *Transformer* module. Two documents which have the same kind of layout are compared by *Information Block Extractor* module and the generated HTML snippet is sent to Translator module. According to the tag mapping table, tags are replaced by specific characters. *Suffix Array Constructor* module generates the repeated patterns hidden in the Web documents. According to the preappointed criteria, *Selector* module finds out the most reasonable one from

aforementioned repeated patterns. Then the properties and their values can be extracted and stored in XML files. Since the names of properties may not be identified by the domain ontologies, based on a *Name Dictionary*, these names are replaced if necessary. Finally, the knowledge can be generated and merged before added into the knowledge base.

4 Web Documents Pre-process

Most Web pages are written using HTML and much of their content is ill-formed. For instance, sometimes there is no close tag corresponding to an open tag. This will bring us troubles in the IE procedure when we need to know the position of interesting information. Therefore, we must first transform original Web documents from ill-formed expression into well-formed expression. XML expression is well-formed and XHTML is the extension of HTML based on XML. Thus we can transform the original HTML documents into XHTML documents preparing for the extraction. Tidy [18] is W3C open source and developed to assist Web authors in improving the accessibility and usability of HTML documents. Here, we use it to do the document transformation from HTML to XHTML.

5 Raw Information Extraction

5.1 Wrappers Generation

Definition 1. *$\mathcal{W}$ is a Web page including the information of interests. A wrappers generation problem is to determine a reflection $\mathcal{R}$ which maps the important information in $\mathcal{W}$ to a data pattern $\mathcal{P}$.*

In Definition 1, $\mathcal{R}$ can identify the Web pages which are similar to $\mathcal{W}$. "Two Web pages are similar" indicates that the layouts or the ways of describing data are similar for both pages, however, their contents are different. Generally speaking, there are two phrases for generating wrappers:

- **Learning**. In this phrase, the patterns of information are extracted from Web pages and then the extraction rules are generated.
- **Management**. It actually indicates the maintenance of the extraction rules.

In this section, in order to construct wrappers, we use a suffix-array-based method for learning patterns from Web documents. Moreover, this self-adaptive method can deal with the updating of Web pages well. Therefore, the management of wrappers is more convenient for users. It is known that a Web documents may contain several information blocks. Some of these blocks contain the information which users are interested in. We recognize this kind of blocks as Information Content Blocks of Interests (ICBI). Other blocks may contain advertisements or other information which are unhelpful to users. Obviously, the repeats that occur regularly and closely in a Web page often represent blocks of meaningful information that might be interesting to users. Here, we use suffix trees and suffix arrays to generat wrappers. A suffix tree, also named PAT tree(or

Patricia tree), is developed by Gonnet to locate every possible position of a prefix in a continuous data stream [21]. In 1993, Manber and Myers proposed the suffix array as a space-efficient substitute for suffix trees [22]. It is simpler and more compact than the suffix tree structure [23]. The suffix array of a string is the array of all its suffixes sorted by lexicographically. It has been used for indexing texts and addressing string problems such as approximate string matching, computing substring frequencies and finding out the longest repeated substrings. Since each Web document can be recognized as a string, we can use a suffix tree to store all the simi-infinite strings(also named sistring) in a document. With the suffix array data structure for implementing the suffix tree, we can analyze each sistring and discover helpful patterns based on specific criteria. Actually, these patterns tell us how to construct wrappers to extract information from Web pages. We give five steps to find out the most reasonable patterns as follows:

Pre-Process of Web Pages. There are some blocks in original HTML documents which do not contain any information, such as *Script* blocks, *Style* blocks and *Comment* blocks. We can first exclude these kinds of blocks and this will benefit later steps. Here, we use regular expressions to match these blocks and then omit them. For instance, we can find out the positions of *Script* blocks using regular expression "$< SCRIPT[\hat{}>] * (> | \setminus n*)$".

Information Block Extraction. The layouts of similar Web pages are same since they are usually generated by an identical Web site. Moreover, the helpful content to us is usually not the same but different in Web pages. Therefore, we can compare every two lines in two similar Web pages from top to bottom. However, we should not omit all the lines which are the same since some of them may belong to ICBI. During the comparison, we denote the line containing the same source codes with letter S(same) and the line containing different source codes with letter D(different). Then we can obtain a string composed of S and D. Based on the above analysis, the number of S is much larger than that of D. The substring containing only one kind of characters is recognized as a line group. If the number of S in a line group exceeds a preappointed threshold (20 characters), the corresponding source codes in the Web page will be omitted. Otherwise, the source codes are retained. Finally, we can get a new string representing the Web document, in which most constant content we are not interested in has been excluded.

Suffix Array Construction. The distribution of tags can reflect the pattern of the information in Web documents. We do not concern about the attributes within tags. Therefore, we omit them and only retain the tag names. Moreover, we replace the literal content between two tags with a new tag </txt>. Then we get a string which contains only tags and based on this string we construct a suffix array to discover patterns. Before constructing the suffix array, we map each tag to an identical letter. Now, we can get a string of letters representing tags in the Web document and do the suffix-array-construction work following steps: **1**. Construct the suffix array of all the suffixes starting at positions i mod $3 \neq 0$; **2**. Construct the suffix array of the remaining suffixes using the result in the first step; **3**. Merge the two suffix arrays into one.

Patterns Discovery. Now, based on the generated suffix array, we expect to find out the maximal repeats hidden in its corresponding Web document. Suppose this array is denoted by SA and has n elements. We define an auxiliary array A with $n+1$ dimensions. $A[i]$ denotes the length of the common maximal suffix of both $SA[i-1]$ and $SA[i]$, where $0 < i < n$. Moreover, the first element $A[0]$ and the last element $A[n]$ are both equal to zero. We also define another array SP to store the serial number of the suffix array SA corresponding to the start position of the common prefix. The algorithm for generating array A and array SP are shown in Algorithm 1. According to array SA and array A, we can get the repeated substrings in the Web document string and by mapping them back to original tags we can obtain hidden patterns. For instance, a substring is a repeated patterns if it starts with the position $SA[i]$ and its length is $A[i+1]$, where $0 \leq i < n$.

Patterns Selection. Since most of the extracted patterns are not incomplete

Algorithm 1 Patterns discovery algorithm.

Input: The suffix array *sa* for a Web document string which is denoted by an array *d*

Output: The maximal common suffixes(patterns)

1. Initialize the auxiliary array a;
2. **for**(i = 0; i < the length of *sa*; i++) {
   2.1 Store the start position of sa_{i-1} of d in $left\_idx$;
   2.2 Store the start position of sa_i of d in $right\_idx$;
   2.3 Initialize a counter: $count = 0$;
   2.4 **while**($left\_idx <$ the length of d && $right\_idx <$ the length of d)
      2.4.1 **if**($d_{left\_idx} == d_{right\_idx}${
         2.4.1.1 $count = count+1$; $left\_idx = left\_idx+1$; $right\_idx = right\_idx+1$;}
            else{
         2.4.1.2 $a_i = count$;**break**;}
   }//generating the auxiliary array A
3. Initialize the serial-number array sp and a stack s;
4. Initialize the two integers: $t = 0, u = 0$;
5. **for**(int i = 0; i < the length of sp+1; i++) {
   5.1. **if**(sp is empty && $a_i > 0$)
      5.1.1. s.$push$(i);
   else{
      5.1.2. Store the top element of s in t;
      5.1.3. **if**($a_t < a_i$)5.1.3.1. Push i into s;
      else if($a_t == a_i$)5.1.3.2. **continue**;
      else{5.1.3.3. $b_u = s.pop()$; $u = u+1$; $i = i-1$;}}
   }//generating the serial-number array SP

or redundant and some of them are even invalid, we should find out the criteria for choosing an outstanding pattern. Here, "outstanding" means that with this pattern we can extract most helpful information from a specific kind of Web

documents. We give our patterns selection method based on some of their features as follows:

- `F1: Tag Loops`. Based on the above pattern discovery method, inevitably, some patterns' tags contain loops. For instance, "<td><txt/></td><td><txt/></td><td>" and "<td><txt/></td>" are equivalent patterns in the information extraction process. This kind of long patterns should be simplified firstly. Algorithm 2 describes our simplification process.

Algorithm 2 Tag loops elimination algorithm.

Input: The string *str* representing a pattern.
Output: The simplified *str*.

1. Initial the position of left pointer(PLP) and the position of right pointer(PRP).
2. **while**(PLP is on the left of PRP){
   2.1 Store the str's substring starting with index 0(included) and ending with index PLP(not included) in string ls;
   2.2 Store the str's substring starting with index PRP(included) and ending with index str.*length*()(not included) in string rs;
   2.3 **if**(ls == rs){
      2.3.1 Store str's substring starting with index 0(included) and ending with index right_point(not included) in str;
      2.3.2 PLP = 1; PRP = str.*length*() - 1;}
      else{
      2.3.3 PLP = PLP + 1; PRP = PRP - 1;}
   }

- `F2: Pattern Length.` Sometimes, the length of the pattern is too short (less than three or four tags). This kind of patterns can not match much of ICBI and the extracted information is usually incomplete. If the length of a pattern dose not exceed the preappointed threshold, we abandon it.
- `F3: First Tag.` A complete and valid pattern should start with an opening tag(like $< TagName >$) or a single tag(like $< TagName/ >$). Therefore, we omit the pattern starting with a closing tags.
- `F4: Coverage.` Some patterns can only cover short parts of the string in ICBI. Here, we give a function to calculate the covering percentage of patterns. Suppose there is a document string S and a candidate pattern P. After matching, there are k break points $point_1$, $point_2$,..., $point_k$. Each break point indicates a start position of the substring which this pattern matches. The covering percentage CP is defined as follows:

$$CP = \frac{k \times Length(P)}{Length(S)} \tag{1}$$

Here, $Length(X)$ denotes the length of string X. If the covering percentage of a pattern exceeds a preappointed threshold, we retain it. Otherwise, we abandon it.

After above selection, we can get a more reasonable pattern for ICBI. Of course, users can modify the pattern manually if necessary. Finally, we use a regular expression to describe this outstanding pattern and recognize it as the wrapper to extract raw data from this kind of Web documents. The properties and their values of extracted individuals are saved in XML files in the end.

6 Knowledge Generation and Merging

6.1 Names of Resources Mapping

The data newly extracted in Section 5 is raw since the names of extracted properties are the original names and a property may have several synonymous names on different Web sites. Therefore, before generating knowledge statements, we should map the original names of properties to the names predefined in domain ontologies. This process is named by *name mapping*.

We establish a *Names Dictionary* (*ND*) to implement the *name mapping* task. Each record appearing in the *ND* denotes a resource (class, property or individual) in the domain ontologies and contains two columns. The resources' names predefined in the domain ontologies are saved in the left column and remarked as *OntName*. On the other hand, the corresponding synonymous names of resources from various Web sites are saved in the right column. As a matter of fact, the synonymous names can be accumulated manually during the IE process if necessary as shown in Figure 1. In the mapping process, the program will check the name of each resource in the XML files generated in section 5. If some resource's name is found in the right column of *ND*, it will be replaced with the name predefined in the domain ontologies. Otherwise, users will be invoked to decide whether the resource corresponding to this new name exists in the domain ontologies or not. If it is a known resource, the name will be recognized as a new synonymous name and added in the right column of the corresponding resource. Moreover, its corresponding *OntName* will replace this new synonymous name in the XML file. Otherwise, the resource does not have any correct corresponding ontology and will be abandoned finally. In order to get high efficiency of consulting *ND*, we use Binary Balance Tree (BBT) data structure to rearrange and store the resources names in *ND* according to their number of characters.

6.2 Knowledge Generation

In this subsection, we use Resource Description Framework (RDF) [19] statements to represent knowledge. Extracting information from Web pages is actually a process of extracting subjects, predicates and objects. Moreover, we find that predicates and their values often appear in structured content by large amount of observations. Therefore, we can identify the subject and the object of each property by querying domain ontologies. Finally, subjects, properties and objects can be described by 3 tuples formation and then added into the

Knowledge Base (KB). In [2], Gruber, coming from Stanford University, given a definition of ontology that an ontology is a specification of a conceptualization, used to help programs and humans share knowledge. It is known that ontologies can be expressed with RDF graphs and we use Jena [20] to query these graphs. Jena takes subjects, predicates and objects as resources. Each predicate has its domain and range: the domain indicates that a particular property applies to a designated class; the range indicates that the values of a particular property are instances of a designated class. Therefore, by querying the domain and range of a specific property, we can identify the classes its subject and object corresponding to respectively. With the wrapper generated in section 5, we can obtain the XML file which contains all the properties and their values in a specific Web document. Before we get the RDF statements, we should identify the types of these properties. Generally speaking, property has two types: Object property and Datatype property. The range of Object property is a class and the range of Datatype property is a literal (string, integer, etc). The types of properties have already been predefined in the domain ontology. Based on the above analysis,

Algorithm 3 Knowledge generation algorithm.

Input: The properties *pro* appearing in the mapped XML files and the domain ontologies *ont*.
Output: The RDF files which contain the extracted knowledge.

1. **for**(i = 0; i < the number of *pro*; i++){
   1.1. Generate a new individual sub_i;
   1.2. **if**(pro_i is an object property){
      1.2.1 Generate a new individual obj using the class of pro_i;
      1.2.2 Add the property pro_i and the value obj to sub_i;}
      else if(pro_i is a data property){
      1.2.3 Add the property pro_i and its value to sub_i;}
   1.3. Link the new individual sub_i to the individuals in *ont*;
   1.4. Merge the new individual sub_i and the individuals in *ont*;
}
2. Return the RDF statements in *ont*;

the properties can be extracted from XML files in the structure in a depth-first search manner. For each node in the XML tree structure, the domain and range of its corresponding property are investigated respectively. Thereafter, we can get all the properties by referring to the domain ontologies and generate knowledge. Algorithm 3 describes the algorithm for generating RDF statements. Note that there may be some relationships among newly generated subjects. For instance, some subjects are redundant resources which should be excluded or some subjects are linked to other subjects by some properties predefined in the domain ontologies.

6.3 Merge Newly Generated Individuals in KB

When a new individual is generated, before adding it into the KB, we should check if the individual has already been included in KB. This process is named as *individual aligning*. It is known that the same individual is usually described in different ways by different Web sites. However, the RDF statement describing an specific individual should be unique in KB. In other words, we should omit the redundant properties of a specific individual.

Suppose ind_A denotes an individual which has already stored in KB and ind_B denotes a newly generated individual. The set of ind_A's properties P_A is denoted by $\{Pind_{A1}, Pind_{A2}, ..., Pind_{An}\}$; the set of ind_B's properties P_B is denoted by $\{Pind_{B1}, Pind_{B2},...,Pind_{Bm}\}$. Here, we recognize the values of properties as strings. Since a string can be recognized as a set of Chinese characters or digits, we calculate the similarity of string α and string β by the function $simSTR(\alpha, \beta)$ defined as follows:

$$simSTR(\alpha, \beta) = \frac{\alpha \bigcap \beta}{min(|\alpha|, |\beta|)} \tag{2}$$

Here, $|\alpha|$, $|\beta|$ denote the length of string α and string β respectively. If the similarity exceeds a preappointed threshold 0.75, two strings are recognized as similar strings. We give the following definition to decide whether two individuals are the same or not.

Definition 2. *ind_A and ind_B indicates equivalent iff their similarity $simIND$ (ind_A, ind_B)$\geq \eta$, where η is a preappointed threshold and*

$$simIND(ind_A, ind_B) = \frac{\sum_{k=1}^{|P_A \bigcap P_B|} simSTR(pv_A(P_k), pv_B(P_k))}{|P_A \bigcap P_B|} \tag{3}$$

Here, $pv_A(P_k)$ denotes the value of a property in ind_A and this property is the kth property in set $P_A \bigcap P_B$; $pv_B(P_k)$ denotes the value of a property in ind_B and this property is the kth property in set $P_A \bigcap P_B$. $|P_A \bigcap P_B|$ denotes the length of set $P_A \bigcap P_B$.

Based on Definition 2, we can decide whether two individuals are equivalent or not. If there is an individual stored in KB satisfying that the similarity between it and the newly generated individual exceeds the threshold η, they are recognized as equivalent individuals and the new properties are added into KB. Otherwise, the newly generated individual will be simply added into KB. In Algorithm 3, the method *mergeIndividual*() is used for implementing the above merging task.

7 Experiments

The experiments for our IE approach are carried out on a PC with an Intel Pentium 1.8GHz CPU and 512MB main memory. In order to provide maximum

transparency of our approach's performance, we gather a sample set of Web documents randomly from Sina, Sohu, Yahoo China, NetEase, Tencent which are large Web sites in Chinese language covering a lot of information about finance, and HengXin, AllInOneNet, HeXun and GuTianXia which are professional Web site about negotiable securities. From each Web site, we retrieve 20 documents. Based on these documents, we expect to extract the information about corporations which have come into the market.

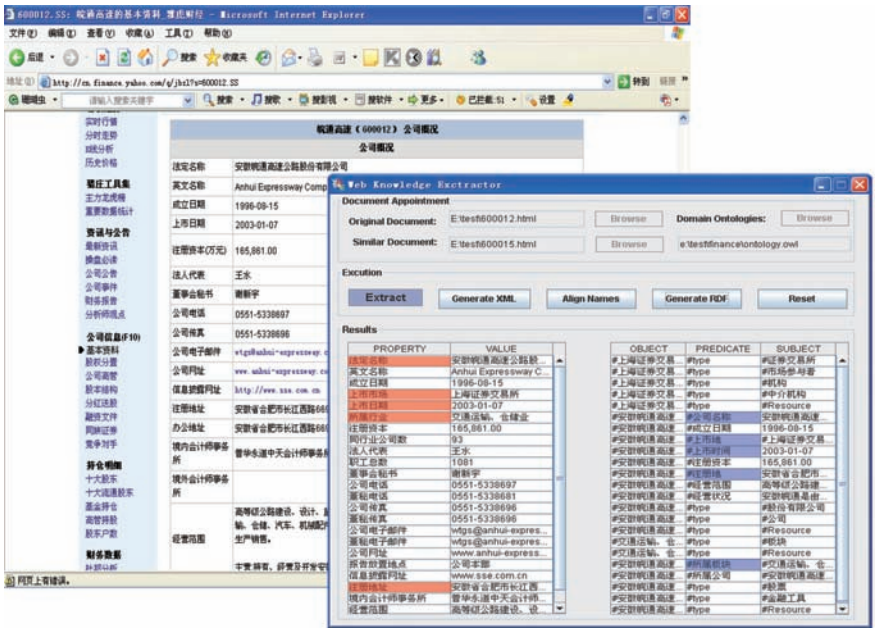

Fig. 2. Excerpt of a Web page

We developed a graphical user interface WebKER (**Web K**nowledge **E**xtracto**R**) for extracting Web knowledge using the Java language. The working process is as follows: users first select the Web document they expect to extract knowledge from and another Web document similar to it. Then, users select the relevant domain ontologies and do the raw-data extraction by pushing the Extract button. Raw data will be displayed in the bottom left table for users' checking. After name alignment, users can get the final extracted knowledge represented with RDF statements and the corresponding subjects, predicates and objects will be displayed in bottom right table. Figure 2 describes an excerpt of a Web page associated with the basic information for "安徽皖通高速公路股份有限公司" (Anhui Expressway Company Limited) from Yahoo China and shows a screen snapshot for our interface when knowledge is extracted from this Web page.

We evaluate the performance of our IE system by calculating *Recall*(R), *Precision*(P) and *F-measure*(F). Suppose that the individuals extracted by experts are sound and complete. *Recall* and *Precision* are defined as follows:

$$R = \frac{|S_{IES} \bigcap S_{DE}|}{|S_{DE}|} \times 100\%, \quad P = \frac{|S_{IES} \bigcap S_{DE}|}{|S_{IES}|} \times 100\% \tag{4}$$

Here, S_{IES} denotes the set of individuals extracted by our IE system and S_{DE} denotes the set of individuals extracted by domain experts manually. The values of *Recall* and *Precision* are between 0 and 1. When *Recall* increases, *Precision* decreases, and verse visa. *F-measure* is another way of evaluating the extraction performance, defined as follows:

$$F = \frac{(\beta^2 + 1)RP}{R + \beta^2 P} \tag{5}$$

Here, β is a preappointed value, which affects the importance degrees of *Recall* and *Precision*.

After normalized, all the "broken" tags in this document are repaired and its corresponding suffix array is established. Then wrappers are generated and used for extracting raw data. We can get the XML file describing the raw information in this Web page. Then based on *ND*, we map the resources' original names to *ONT names* and get a new version of the XML file describing this corporation. The resources' original names which are not included in *ND* are not changed after mapping. Moreover, these resources will be automatically abandoned within the RDF-statements-generation process. See the interface snapshot in Figure 2, the names with red backgrounds in the bottom left table are replaced with the ones with blue backgrounds in the bottom right table.

Now, the mapped XML file can be used for generating knowledge based on the domain ontologies. According to Algorithm 3 described in subsection 6.2, the individuals can be extracted from the Web document and stored in RDF file. After merging, these individuals can be added in KB directly, since the knowledge in KB is also described by RDF statements. Finally, *Recalls*, *Precisions* and *F-measures*($\beta = 1$) of extracting the information from the aforementioned Web sites by using our IE approach are shown in Table 1. From Table 1, we can see that for most Web sites, *Precision*s, *Recall*s and *F-measure*s are satisfactory using our system. The extraction from NetEase fails, since the content of ICBI in the documents on this Web site is generated dynamically with JavaScript. Therefore, in this case, our system is helpless. The recall for GuTianXia is lower than other Web sites' since a specific kind of individuals is missed in all pages on this Web site. However, the Table 1 indicates that our approach can be applied to most of Web sites well.

Traditional methods store the information extracted by wrappers in a database for users's queries. This can be easily implemented through saving the generated XML files into a relation database. Here, we compare the performance of querying the database and that of querying our knowledge base by calculating *Recall*, *Precision* and *F-measure*($\beta = 1$). We generate 120 queries to do our experiments

Table 1. Performance for individual extraction

	Web Sites	URL	Individuals Extracted By DE	Performance Cor.	Err.	Prec.	Rec.	F1
S1	Sina	www.sina.com	141	141	19	88.13%	100%	93.69%
S2	Sohu	www.sohu.com	100	100	20	80.00%	100%	88.89%
S3	Yahoo China	www.yahoo.com.cn	125	125	15	89.29%	100%	94.34%
S4	NetEase	www.163.com	120	0	0	0%	0%	0%
S5	Tencent	www.qq.com	58	58	22	65.00%	100%	78.79%
S6	HengXin	www.hengx.com	99	99	1	99.00%	100%	99.50%
S7	AllInOneNet	info.cmbchina.com	71	71	9	86.25%	100%	92.62%
S8	HeXun	www.hexun.com	100	100	0	100%	100%	100%
S9	GuTianXia	www.gutx.com	80	60	0	100%	75.00%	85.71%

focusing on five Web sites. Table 2 shows the results of this comparison. In this table, we can see that the *Recall*s of querying knowledge base are all higher comparing to those of querying traditional data base. The higher *F-measure*s indicate the superiority of our domain-ontology-based IE approach.

Table 2. Querying database versus querying knowledge base

	Web Sites	Ques.	Performance For DB					Performance For KB				
			Cor.	Err.	Prec.	Rec.	F1	Cor.	Err.	Prec.	Rec.	F1
Q1	Sina	120	67	9	88.16%	55.83%	68.37%	91	15	85.85%	75.83%	80.53%
Q2	Sohu	120	72	14	83.72%	60.00%	69.9%	99	0	100%	82.50%	90.41%
Q3	Yahoo China	120	57	7	89.06%	47.50%	61.96%	88	20	81.48%	73.33%	77.19%
Q4	HengXin	120	49	1	98.00%	40.83%	57.64%	119	0	100%	99.17%	100%
Q5	HeXun	120	52	1	98.11%	43.33%	60.11%	120	0	100%	100%	99.58%

8 Conclusions

In this paper we propose an information extraction approach based on suffix arrays and domain ontologies. It can automatically learn wrappers from Web documents and generate knowledge for users to easily query. Experiments on documents from large Chinese Web sites yielded very promising results. Our current research goal focuses on improving our wrapper generation module by adding heuristics in the pattern-discovery process and the pattern-selection process. The performance of our approach is based on the soundness and the completeness of the interrelated domain ontologies, and our long term goal is to automatically or semiautomatically extend the domain ontologies dynamically based on the raw information extracted with our wrappers.

References

1. J. Cowie and W. Lehnert, *Information Extraction*, Communications of the ACM, Vol. 39, No. 1, pp. 80-91, 1996.
2. T. R. Gruber, *A translation approach to portable ontology specifications*, Knowledge Acquisition, Vol. 5, No. 2, pp. 199-220, 1993.
3. D. Florescu, A.Y. Levy and A.O. Mendelzon, *Database techniques for the world-wide web: A survey*, SIGMOD Record, Vol. 27, No. 3, pp. 59-74, 1998.
4. S. Soderland, *Learning to extract text-based information from the world wide web*, In Proceedings of the KDDM'97, 1997.
5. N. Kushmerick, *Wrapper induction for information extraction*, PH.D. Dissertation, University of Washington, Tech Report UW-CSE-97-11-04.
6. S. Chawathe, H. G. Molina, J. Hammer et al, *The TSIMMIS project: Integration of heterogeneous information sources*, Proceedings of the IPSJ'94, pp. 7-18, October 1994.
7. I. Muslea, S. Minton and C. Knoblock, *A hierarchical approach to wrapper induction*, Proceedings of the AA'09, pp. 190-197, 1999.
8. S. Soderland, *Learning information extraction rules for semi-structured and free text*, Machine Learning, vol. 34, pp. 233-272, 1999.
9. Y. Jaeyoung, O. Heekuck, D. Kyunggoo and C. Joongmin, *A Knowledge-Based Information Extraction System for Semi-structured Labeled Documents*, Proceedings of IDEAL'02, LNCS 2412, pp. 105-110, Springer Verlag, 2002.
10. B. Habegger and M. Quafafou, *WetDL: A web information extraction language*, Advances in Information Systems, LNCS 3261, pp. 128-138, Springer Verlag, 2004.
11. E. H. Pek, X. Li and Y. Z. Liu, *Web wrapper validation*, Proceedings of the APWeb'03, LNCS 2642, pp. 388-393, Springer Verlag, 2003.
12. B. Chidlovskii, J. Ragetli and M. D. Rijke, *Wrapper generation via grammar induction*, Proceedings of the ECML'00, LNCS 1810, pp. 96-108, Springer Verlag, 2000.
13. M. Gatius and H. Rodríguez, *Natural language guided dialogues for accessing the web*, Proceedings of the TSD'02, LNCS 2448, pp. 373-380, Springer Verlag, 2002.
14. B. Habegger and D. Debarbieux, *Integrating data from the web by machine-learning tree-pattern queries*, Proceedings of the ODBASE'06, LNCS 4275, pp. 941-948, Springer Verlag, 2006.
15. X. B. Deng and Y. Y. Zhu, *L-Tree match: A new data extraction model and algorithm for huge text stream with noises*, Computer Science and Technology, No. 20, Vol. 6, pp. 763-773, Springer Boston, 2006.
16. C. Schindler, P. Arya, A. Rath and W. Slany, *htmlButler –wrapper usability enhancement through ontology sharing and large scale cooperation*, Adaptive and Personalized Semantic Web, Vol. 14, pp. 85-94, Springer Berlin, 2006.
17. D. D. Lewis, *Naive bayes at forty: the independence assumption in information retrieval*, Proceedings of the 10th European Conference on Machine Learning, LNCS 1398, pp. 4-5, Springer Verlag, April 1998.
18. HTML Tidy Project, http://tidy.sourceforge.net
19. RDF Primier, http://www.w3.org/TR/rdf-primer
20. Jena APIs, http://jena.sourceforge.net/downloads.html
21. G. H. Gonnet, R. A. Baeza-yates and T. snider, New in dices for text: Pat trees and Pat Arrays, Information Retrieval: Data Structures and Algorithms, Prentice Hall, 1992.

22. U. Manber and G. Myers, Suffix arrays: a new method for on-line search, SIAM Journal on Computing, No. 22, pp. 935-948, 1993.
23. M. I. Abouelhoda, E. Ohlebusch and S. Kurtz, Optimal exact string matching based on suffix arrays. Proceedings of the 9th Symposium on String Processing and Information Retrieval, LNCS 2476, pp. 31-34, Springer Verlag, 2002.