

Identification of Query Forms for Retrieving the Information from Deep Web

¹Nripendra Narayan Das and ²Ela Kumar

¹Rawal Institute of Engineering and Technology, India

²IGDT University for Women, India

¹nripendradas@gmail.com; ²ela_kumar@rediffmail.com

ABSTRACT

Web databases are now present everywhere. The data from the Deep Web cannot be accessed by Search engine and web crawlers directly. The only way to access the hidden database is through query interfaces and filling up number of HTML forms for a specific domain [18]. In this paper a technique called QFORT (QUERY FORM RETRIEVAL TECHNIQUE) has been developed for identifying the relevant query form is presented. Retrieving information from deep web pages using wrappers is a fundamental problem arising in a huge range of web pages of vast practical interests. In this paper, we propose a novel technique to the problem of identifying the query forms from Web pages, which is one of the key problems in automatic extraction approach. The problem is resolved by many authors by using different technique 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.

Keywords: Crawler, deep web, wrapper generation, attribute.

1 Introduction

Information searching has been becoming one of the most important and popular activities on the Web. Today's web based crawler retrieve content only from a portion of the web called the publicly indexable web (PIW) [1] i.e. the set of web pages which can be seen purely by following hypertext links, by ignoring search forms and the pages which require user authorization or registration. During studies it was observed that significant fraction of web content lies outside the PIW [18]. It is estimated that there are several million hidden-web sites [2] with various important information whose results are hidden behind the search form. These types of web are called the hidden web or deep web [2]. Pages in the hidden web are dynamically extracted after submitting the query through different search forms.

The great challenge with the researchers are to identify and automatically fill these search forms. As all the pages, which contain the search forms, include at least one HTML form which is used to submit the query parameters. In this paper a technique called QFORT (QUERY FORM RETRIEVAL TECHNIQUE) has been developed for identifying the relevant query form is presented.

Extracting information from the deep web can be categorized as follows:

- (1) There must be detail description of search method.
- (2) Searching of relevant query forms which are relevant to the task.

(3) Search form must be filled up and examine the results of each relevant and useful information.

(4) Query must be formulized.

It is very much challenging for user to retrieve and analyze relevant as well as important information from the deep web automatically. Now a days, the users manually fills input values to web forms and extract data from the returned web pages. In case of user wish to fill complex queries, filling out forms manually is not practical, but these queries are required for many web based applications [18]. In this paper we are concentrating only on one specific domain i.e. Automobile. While browsing different types of automobile web site, it was observed that the web sites are designed in same pattern and users need to fill up number of consecutive forms to retrieve the information. It is explained with an example.

Example 1: As explained above it was observed that all car search related web sites are designed using the same model. As shown in figure-1 the first web page consists of the these fields (i) Make field (ii) Radio/Checkbox button with “New car” and “Used cars (iii) Name of state / city / pin code and (iv) Search button etc. etc.

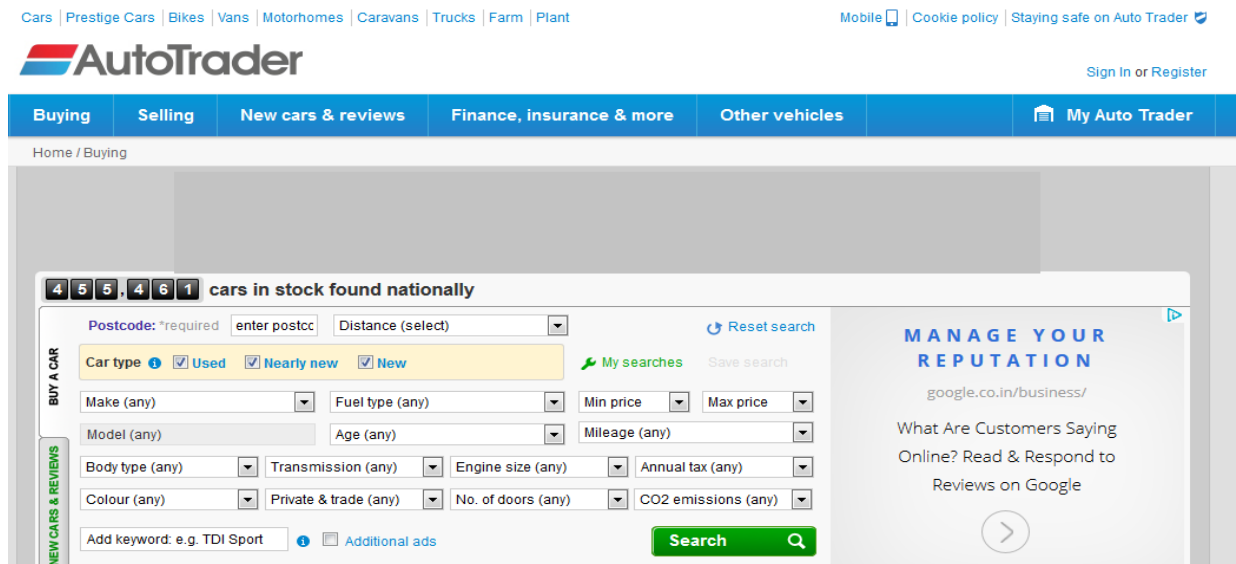


Figure 1: Car Search Form

1.1 Site Form Analysis:

It has been assumed that the given HTML page has a form that applies to our chosen application. In the case when more than one form is on the page, we consider only the largest form—the one with the largest number of characters between the open and closing form tags. Its content are then parsed into a DOM tree.

Form designers create many fields with input tags. For example:

```
<input type="text" size="10" name="zip" maxlength="5" value="">
```

They use select and textarea to create other fields. An example is:

```
<form method="post" class="searchForm" action="http://www.autotrader.co.uk/search/form">
```

```
<select name="radius" id="radius" class="searchFilter">
```

```

<option value="1500">Distance (select) </option>
<option value="1">within 1 mile</option>
-----
-----
-----
<option value="200">within 200 miles</option>
<option value="1501">National</option>
</select>
</p>
<option class="" value="" >Make (any) </option><option class="" value="abarth" >ABARTH (42)
</option>
-----
-----
-----
<option class="" value="ac" >AC (24) </option>
</select>
<option class="" value="" >Min price </option><option class="" value="0" >Rs. 0
</option><option class="" value="500" >Rs. 5,00,000 </option>
<option class="" value="1000" >Rs. 1,00,000 </option>
-----
-----
-----
<option class="" value="1500" >Rs. 15,00,000 </option>
<option class="" value="2000" >Rs. 20,00,000 </option>
<option class="" value="" >Mileage (any) </option>
<option class="" value="up_to_100_miles" >up to 100 miles (17037) </option>
-----
-----
-----
<option class="" value="up_to_1000_miles" >up to 1,000 miles (22049) </option>
-----
-----
-----
</form>

```

Figure 2: Equivalent codes of Figure-1

Although there are many attributes for the input tag, we are only interested in the type, name, and value attributes. After parsing the input tag, we store the field name, field type, and field value for fields with type text, hidden, checkbox, radio, and submit. For the text area tag, we store the field name with

field type text area. For the select tag, we analyze the content between the opening and closing tags to extract and store the field name, the option values (values inside the option tags), and the displayed values (values displayed in the selection list on the Web page).

User may give any type of unexpected query in search engine text box e.g. suppose the user wants to find information about the “used” Japanese red cars of 2006 made within Rs. 15 Lakh and available in “Delhi”. User will have to fill up the form shown in figure 1 to formulate such types of query by selecting the choice in make field (e.g. Honda, Toyota etc.). More importantly, User has to fill up and submits the form repeatedly for all different Japanese car. As most of the web pages do not contain colour option, user has to browse each and every result for a particular colour car.

The above process can be efficiently completed by using an automatic form querying system, but it is not an easy task to design this type of automated query processing method due to several challenges.

The challenges are as follows:

- (a) Automatic filling of forms: As web pages provides different types of interfaces, automatic filling of forms is a challenging task. Moreover, the user may not be aware of some of the important required field which may be mandatory field for some web site. (E.g. Filling of PIN code to find out the city name is a difficult task for user).
- (b) Extraction of results: As most of the data displayed in result pages of web site are embedded in HTML code and this is another difficult problem to extract the result from the web pages. The search and the extraction of required data from such pages are very much complicated task since each web form interface is designed for user’s convenient and every web page format are always different from each other.
- (c) Navigational complexity: The pages which are generated after submission of query form may contain link to another web pages consists of relevant informations and therefore, it is necessary to navigate these links to see the detail record. It was also observed that during navigation of such web sites repeated filling of web forms are required which are dynamically generated by the server side programs due to submission of pervious query form. These forms are collectively called consecutive forms.

In this paper, a query technique is discussed for the deep web called QFORT (QUERY FORM RETRIEVAL TECHNIQUE) and resolve above mentioned challenges.

To solve the above issues following steps are required:

- (a) *Identification of relevant forms for specific domain database*
- (b) *Automatic filling up of forms with the keywords and display the results.*

The rest of the paper is organized as follows: Section 2 discusses related research in this area. Section 3 and 4 present model for representing normalized forms.

2 Related Work

The huge growth of the deep web has motivated interest in the study of web crawlers [3, 4]. Currently, many research works are going on for the extraction of information from deep web in better way and many solution have been proposed by the researchers. Some of important types are:

- Manual Approach
- Wrapper generation
- Automatic extraction.

Manual Approach: In this approach programmer tries to extract the information from web pages after identifying it's schema by writing equivalent source code.

Wrapper generation [5]: In this approach, set of rules are designed to extract the information from web pages.

Automatic Extraction [6]: Data items have different meaning and role in web pages. In this approach authors have proposed a method to identify the mapping between data items having same role in different pages.

3 Searching of Single Html Form

In this section, the attribute-value representation is discussed in a single HTML form. It was observed that the nature and type of the layout markup are not same in HTML forms. The web application must follow the uniformity in designing the form.

3.1 HTML forms:

An HTML form consists of various parameters as per the requirement e.g. textboxes, dropdown menus, radio buttons, check boxes including banners, advertisement, diagram etc... After filling up all the required information available in forms user submit the form to web server or mail server for further processing. HTML form are designed with the HTML code and a form is embedded inside the <FORM> tag. Each HTML form contains a set of form field and the URL address of server side program so that whenever user submits the query form, server returns a set of result pages.

To process a FORM in server side mainly three essential attributes are required:

- Action attribute – It contains the URL address of the form.
- Method attribute – HTTP method is used to submit the form.
- Enctype attribute – It specify the content type used for form processing.

3.2 Searching of relevant query forms [18]:

In the previous section it was discussed how to design a single form. Sometime the desired results were retrieved by using only a single form called root form. But it was observed in some forms(e.g. auto-search form, book-search form etc..) that user need to fill up more than one form known as the child form(s) or descendant form(s) to retrieve the desired result as shown in figure 1 because the root form has dependency on child form.

Example : As shown in figure-1, if the user wants to search for different model of Maruti make, in this case user has to select the Maruti make in choice make field and after submitting this root form the second form will display all models of Maruti make as shown in figure 1(b).

3.3 Issues in modelling of consecutive forms [18]

Some of the issues were observed in modelling consecutive forms. Most consecutive forms have dependencies between the main and descendant form.

Example: In search interface form as shown in figure-1, user requires two consecutive forms to be fill out, if user searched for “Used Car” & “New Car”. In all existing search forms, it was also observed that there is form dependency exists between form fields. For example, if user selects the “Maruti” in make menu, the all models of “Maruti” make will be displayed in its child form in drop down menu option and if the user wants to see the details of all model one by one he has to select one model at a time and submit the query. It means if there are 10 models available for “Maruti” make (in our example), user has to select 10 times and submit the query to retrieve the results of different model, which is tedious as well as time consuming job.

4 Comparison of PIW Crawler and HW Crawler

In the traditional PIW crawler the basic idea is to follow different web pages from one to another using links. Another way to think it is that the Web is a large directed graph and that the PIW Crawler is simply exploring the graph using a graph traversal algorithm.

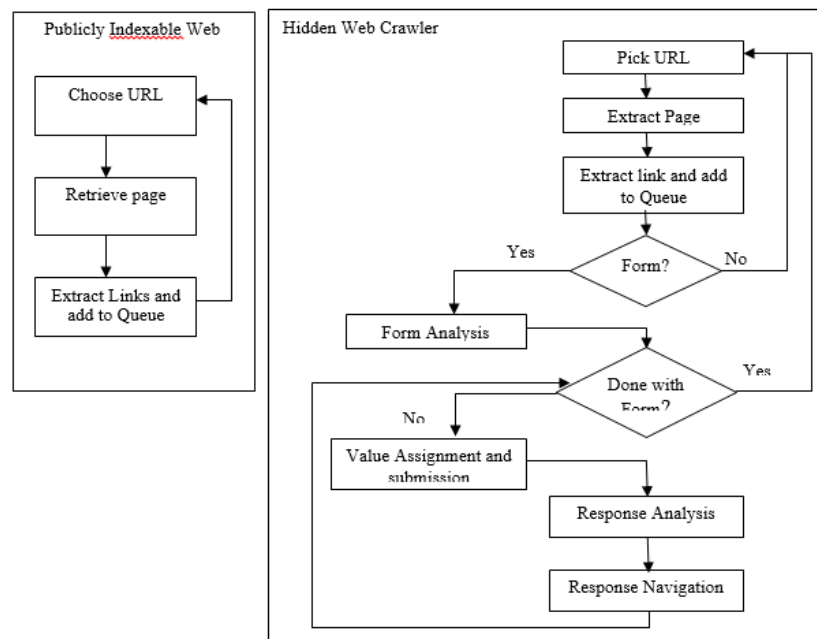


Figure 3: Flow chart comparison of PIW crawler and HW crawler

4.1 Algorithm for Data Extraction from Web pages:

BEGIN

- (i) Insert the keyword in search box (e.g. HONDA city car price 5 lac onwards).
- (ii) Extract the one key word from the number of keywords user entered in the search box.
- (iii) Search the web addresses stored in database (The database store the web addresses of different domain).
- (iv) if keyword matches with the keyword of web addresses, the web page will be opened internally and the corresponding results will be extracted from the that particular web page and display the result. Likewise software will start searching on remaining web addresses and extract the results accordingly.

END

5 Observation

It was observed that in existing form query interface, user fill up the keyword in search box and after clicking the submit/search button server sends the results as web addresses. And suppose user is not satisfied with the result of that web page he/she has to open another web link and compare the results as per his/her requirement and he/she will keep on browsing the link unless and until he/she does not get satisfied with the results. But in our approach user need to fill the query in search box in the beginning only and after that crawler will search in the database equivalent web address, open it internally and extract the desired result from that web site and it follow the same procedure for other web addresses also.

6 Experimental Results and Analysis

In this paper, an experimented was conducted on ten Web sites for each of two applications: car ads and Hotel Booking ads. The approach, however, is not limited to the two applications on which experiment was conducted. It can work with other applications as long as those applications have Web sites with forms, and we have ontologies for those applications. The process of rewriting queries in terms of site forms is the same.

6.1 Experimental Results

We are interested in three kinds of measurements: field-matching efficiency, query submission efficiency, and post-processing efficiency.

To know if we properly matched the fields in a user query with the fields in a site query, we measured the ratio of the number of correctly matched fields to the total number of fields that could have been matched (a recall ratio for field matching (f_m)), and we measure the ratio of the number of correctly matched fields to the number of correctly matched fields plus the number of incorrectly matched fields (a precision ratio for field matching (f_m)):

$$\text{Recall}_{f_m} = \frac{\text{Number of correctly matched fields}}{\text{Total number of fields that should have been matched}} \quad \text{Precision}_{f_m} = \frac{\text{Total number of matched fields}}{\text{Number of correctly matched fields}}$$

To know if we submitted the query effectively, we measure the ratio of the number of correct queries submitted to the number of queries that should have been submitted (a recall ratio R_{qs} for query submission, qs), and we measure the ratio of the number of correct system queries submitted to the number of correct queries submitted plus the number of incorrectly submitted queries (a precision ratio P_{qs}):

$$\text{Recall}_{qs} = \frac{\text{Number of correct queries submitted}}{\text{Total number of queries that should have been submitted}} \quad \text{Precision}_{qs} = \frac{\text{Number of correct queries submitted}}{\text{Total number of queries submitted}}$$

An overall efficiency measurement was also conducted which was obtain by multiplying the three recall measurements and the three precision measurements together:

$$R_{\text{overall}} = R_{f_m} * R_{qs}$$

$$P_{\text{overall}} = P_{f_m} * P_{qs}$$

Because the two kinds of metrics measure two stages of one single process, we used the products to calculate the overall performance of the process with respect to our extraction ontology.

Table 1: Experiment result of used car search

Number of Forms : 10 Number of Fields Forms: 42 Number of Fields applicable to the ontology: 37(78.5%)			
	Field Matching	Query Submission	Overall
Recall	100% (37/37)	100% (325/325)	100%
Precision	100% (37/37)	85.5% (325/380)	85.5%

Table 2: Experiment result of Hotel Booking search

Number of Forms : 10 Number of Fields Forms: 25 Number of Fields applicable to the ontology: 18(72%)			
	Field Matching	Query Submission	Overall
Recall	94% (17/18)	100% (212/212)	94%
Precision	100% (18/18)	212% (212/212)	100%

7 Conclusion

In this research, a system is designed and implemented that can fill out and submit Web forms automatically according to a given user query against a corresponding application extraction ontology. From the returned results, the system extracts information from the pages, puts the extracted records in a database, and queries the database with the original user query to get, to the extent possible, just the relevant data behind these Web forms.

Our system has been tested on two applications: car advertisements and Hotel Booking advertisements. In average, there were 78.9% fields in the site forms that were applicable to the extraction ontologies. The system correctly matched 95.7% of them. Considering only the fields that were applicable to the extraction ontologies and were correctly matched, the system correctly sent out all queries that should have been submitted to the Web sites we tested. It, however, also sent out some additional queries that are not necessary according to the original user query. Among all queries our system submitted for our experiments, only 91.4% of them are necessary. Further, for the Web sites we tested, our Output Analyzer correctly gathered all linked pages. Finally, of the records correctly extracted by Ontos, our system always correctly returned just those records that satisfied the user-specified search criteria.

REFERENCES

- [1]. S. Lawrence, C. L. Giles, Searching the world wide web, Science 280(5360)(1998) 98-100. M. K. Bergman, The Deep Web: Surfacing Hidden Value, September 2001, <http://www.brightplanet.com/deepcontent/tutorials/deepwebwhitepaper.pdf>.
- [2]. S. Chakrabarti, M. van den Berg, B. Dom, Focused Crawling: A new approach to topic specific web resource discovery, in 8th World Wide Web conference may 1999.
- [3]. J. Cho, H. Gracia-Molina. The evolution of the web and implication for an incremental crawler, in: Proc, 26th Int. Conf. Very Large Data Bases. VLDB, 2000.

- [4]. Yanhong Zhai and Bing Liu, "Extracting Web Data Using Instances Based Learning." WISE conference, 2005.
- [5]. Hu D, Meng X, "Automatic Data Extraction from Data-Rich Web Pages", The 10th Data System for Advanced Applications (DASFAA), Beijing, 2005.
- [6]. LIU Wei, MENG Xiao-Feng, MENG Wei-Yi, A Survey of Deep Web Data Integration, Chinese Journal of Computers, Vol. 30, No. 9, Sept 2007, pp : 1475-1489.
- [7]. B. He, Patel M, Zhang Z, C Chang, Accessing the deep web, Communication of the ACM, Col. 50 (5), May 2007; 95-101.
- [8]. Yoo Jung An, James Geller, Yi-Ta Wu, Soon Ae Chun: Semantic deep web: automatic attribute extraction from the deep web data source. In proceeding of the 2007 ACM symposium on Applied Computing (SAC2007), Seoul, Korea, March, 2007, pp: 1667-1672.
- [9]. Longzhuang Li, Y onghuai Liu, Abel Obregon, and Matt A.Weatherston "Visual Segmentation-Based Data Record Extraction from Web Documents," IEEE IRJ 2007.
- [10]. Wei Liu, Xiaofeng Meng, and Weiyi Meng, "ViDE: A Vision-based Approach for Deep Web Data Extraction," IEEE TKDE, 2009.
- [11]. Hongkun Zhao, Weiyi Meng, Zonghuan Wu, Vijay Raghavan, and Clement Yu, "Fully automatic wrapper generation for search engines," in ACM WWW, 2005.
- [12]. Wu, W., Doan, A., Yu, C.: WebIQ: Learning from the Web to match Deep-Web query interfaces. In: Proceedings of the 22nd IEEE International Conference on Data Engineering (ICDE 2006), p. 44 (2006).
- [13]. B. He, Z. Zhang, and K. C.-C. Chang. Knocking the door to the deep web: Integrating web query interfaces. In SIGMOD Conference, System Demonstration, 2004.
- [14]. L. Barbosa and J. Freire. Siphoning hidden-web data through keyword-based interfaces. In SBBD, 2004.
- [15]. A. Ntoulas, P. Zerfos, and J. Cho. Downloading hidden web content. Technical report, UCLA, 2004.
- [16]. J. Cope, N. Craswell, and D. Hawking. Automated Discovery of Search Interfaces on the Web. In Proc. of ADC, pages 181–189, 2003.
- [17]. Das N. N., Kumar Ela, Hidden Web Query Technique for Extracting the Data from Deep Web Data Base, WCECS2012_PP410-414, in proceeding of the world congress on engineering and computer science 2012 vol-1 WCECS October 24-26 2012