Design of Hidden Web Search Engine

Anuradha
Department of Computer Engineering
YMCA University of Sc. & Technology
Faridabad, India

A.K.Sharma
Department of Computer Engineering
YMCA University of Sc. & Technology
Faridabad, India

ABSTRACT
Extracting Hidden Web data arises several challenges especially in case of data sources with multi-attribute interfaces wherein multiple attributes and their respective values should be extracted, indexed and stored. Hence, accessing the hidden web content is a potential research area as the pages are dynamically created through search query interface. However, direct query through search interface is laborious way to search. Hence, there has been increased interest in retrieval and integration of hidden web data with a view to give high quality information to the web user or we can say it is highly desirable to build a hidden web search engine. This paper proposes a novel approach for building a hidden web search engine which fills the search forms automatically, extracts the result records, store them in a repository for later searching. This proposed work also provides the user with user interface in which user can fill the query and finds the desired results.

Keywords
WWW, Hidden Web, Information Extraction, Search interfaces, DOM.

1. INTRODUCTION
In the recent years, the World Wide Web has been enriched by the enormous amount of databases on the Internet: While the "Surface Web" has millions of static linked HTML pages, it is believed that a far more significant amount of information is "hidden" in the online databases, behind the query search forms of searchable databases [8]. This information cannot be accessed directly through static URL links, they are only available as responses to dynamic queries submitted through the search query interface of a database. Therefore, this information is hidden behind such forms. Hence, these databases are often also referred to as the "Hidden Web".

Traditional crawlers wander throughout the web and index the addresses of pages they discover. When these software programs run into a page from the Hidden Web, they don't know what to do with it. These crawlers can record the address, but can't deal with the information that page contain [3]. Because current crawlers cannot effectively query these databases, such data remains invisible to traditional search engines, and thus remain largely hidden from users.

Search query forms and dynamically generated pages are helpful to users because users can get the desired information they want [15]. However, it is really tiresome task for users to visit all the hidden web sites for the same domain and fill out different forms provided by each site. Hence, development of an automated process that can fill the search query forms, extracts the results, integrates them, stores them at one location and searches the result that matches user query is highly desirable.

To solve all the problems mentioned above, "A Hidden Web Crawler Based Search Engine" is proposed in this paper that fills out search forms automatically and extracts the relevant data behind those search forms. The architecture of proposed prototype is explained in next section. The paper has been organized as follows: Section 2 describes the related work in the area of Hidden web search engine. Section 3 describes the proposed work in detail. Section 4 discusses the experimental results and finally conclusion of the research is described in section 5.

2. RELATED WORK
This section discusses the work that is closely related to the proposed work. Although there are many proposed hidden web crawling techniques, a very little work is done on the indexing and searching part (i.e search engine) of hidden web data.

HiddenSeek [10] uses a keyword based indexing and searching technique for single-attribute hidden web sites. This approach uses the inverted index for indexing and searching method the hidden web data. HiddenSeek takes a term frequency of keyword as a factor for ranking the results i.e, whether the keyword appears in the URL of a page.

[15] provided a theoretical framework to investigate the query generation problem for the Hidden Web and proposed effective policies for generating queries automatically. This research works for the Hidden websites with interface containing single keyword.

Siphon++ [9] proposed a strategy for automatically retrieving data hidden behind keyword-based form interfaces. It considers the query generation and selection techniques by detecting features of the index. Unlike multi-attribute forms, keyword-based interfaces are simple to query, since they do not require detailed knowledge of the schema or structure of the underlying data. It is thus easy to create automatic and effective solutions to crawl these interfaces. This research also takes single keyword interface into consideration.

[28] states that there are a growing number of pages that can't be indexed by search engines and stay invisible to other surfers, despite the fact that they contain a lot of relevant content. Pages with dynamic content i.e the pages that are the result of a submitted query and consequently do not have a static URL can be impossible for search engines to find, since the crawlers cannot replicate the query submission carried out by human beings. Generally traditional search engines follow links to the index page on a site and then crawl from there to other pages by following links. Search engine crawlers will therefore have more difficulties seeing a page that is not linked to from any other page (Hidden web pages).

3. PROPOSED WORK

The basic idea behind the designing such an engine is to find several different web pages within the same domain like book domain, car domain etc., collect this information and after some compilation and normalization present it as a free web search service.

3.1 Proposed Architecture of Hidden Web Search Engine:

The whole process of the proposed system is divided into two main components and each component requires individual consideration. These components are: Search query Interface Processing & Form Submission and Hidden web Data Extraction, Integration and Searching.

The architecture of Hidden web search engine is shown below in figure 1.

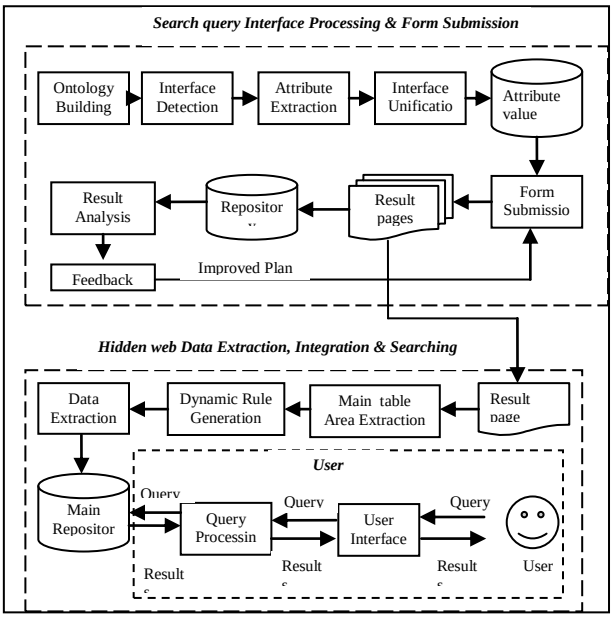

Fig 1: Architecture of Hidden Web Search Engine

The first component interacts with search interface, fills the search form and extracts the results. These results pages are sent to the second phase. The second component processes the result pages one by one, takes out data from the tables of result page and collects them in one place. The details discussion of both the phases is given in next section.

3.2 Search Query Interface Processing & Form submission:

This component of the Search engine deals with the search query interfaces. Search interface represents a subset of queries that could be performed on the underlying Hidden web database. User's query is translated to a structured format, such as SQL, and is executed against the online database. Search query interface acts as entry point to the Hidden web which contains substantial amount of high quality structured data. So, to get the entry inside the Hidden web, search interfaces are explored first.

3.2.1 Proposed Architecture :

A search query interface allows a user to search some set of items without altering them. The user enters a query, by typing or selecting options, to describe the items of interest. Results might be a page linking to items, a page containing items or a single page. The items found should match the query. The majority of search interfaces on the Web are HTML forms. Therefore the problem is: given a URL List, find all HTML search interfaces (search forms). This component of the Hidden Web Search Engine automatically detects the domain specific search interfaces by looking for domain ontology in the source code of the web page.

3.2.1.1 Building Domain Ontology:

Domain ontology can be defined as information architecture in a specific domain. It provides a structured way of describing knowledge. It also defines the concepts about Web page categories and their hierarchical relationships. Ontology is composed of concepts, attributes and the relation among concepts. Concept is anything that can be described. It can be a real, assumed, concrete or abstract. A concept in ontology can be described by the attribute values. The main function of ontology is to provide the knowledge base needed for the classification of search results. Organizing search results into hierarchical structure can help users navigate, seek and find more quickly information they are looking for [23]. Since Hidden Web is a huge repository for all domains, we confined our research to car domain. To build the ontology in this domain, used car keyword is pushed into the search box of Google and result index is extracted. Each result index is downloaded and attributes are extracted.

3.2.1.1.1 Attribute Extraction:

Out of large set of results, 4 sites are taken randomly as sample documents. These sites are downloaded to extract the search attributes. Algorithm for extracting the attributes is given below in figure 2.

```
Algorithm1 atrr_extract(URL List)

1. Alist=φ and i=0;
2. Pick URL one by one from URL List
2. x=document.getElementsByTagName("select");      //
reach upto the point where select is
                                                    present.
3.Repeat steps 4 and 5  while i<  x.length
4.  document.write(x[i].innerHTML);     // it will write the
inner text which is not in <, > tags.
5. Alist =Alist ⋃ x ;
6. end.
```

Fig 2 : Algorithm for Attribute Extraction

The output of this phase is given in table 1.

Table 1: Extracted Atrribute list

Website	Attribute list
www.autonagar.com	make, model, city
www.carwale.com	make, model, city, price
www.gaadi.com	Make, model, budget, city
www.cartradeindia.com	Brand, model, state, city

3.2.2.2 Using Wordnet:

WordNet is a lexical database for the English language [22]. It groups English words into sets of synonyms called *synsets*, provides short, general definitions, and records the various semantic relations between these synonym sets.To make the onology, we have used this dictionary by which synonyms are extracted and a list is made for each attribute. For make, the synonyms are (brand, shuffle) .Similarly for model, synonyms are (framework, example, role model, poser etc.), for price list is (Price, monetary value, cost) and for city (metropolis, urban centre) is the list.

3.2.2.3 Ontology Table:

To make the domain ontology repository, synonyms from the wordnet dictionary are collected along with the attributes from all 4 sites and are stored in database as shown in Table 2.

Table 2 :Ontology table

Atrribute(ai)	Synlist
Make	Select make, brand
Model	Model, framework, example, poser
City	City, cities, metropolis
Price	Price, range, budget

3.2.2.4 Interface Detection:

The algorithm for Interface detection is given below in figure 3. This algorithm works by matching the interfaces with its domain ontology. It checks domain word, its synonyms, attributes and their synonyms in source code of the webpage. If they are present this means these are interfaces of our domain.

```
Algorithm  Interf_detect2(URL List)
1.  Pick one by one URL from URL list
2.  Download the source code and save.
3.  Check the source code for tag <form> and </form>
// Form analyses
    if (present) Goto step 4
    Else  " It is a simple web page".  Goto step 1.
4.  Check  source code
                // Search interface analysis
    if ( input-type="login" or "registration" or" sign
up")or (having password control )
    then      Goto step 1. Else Goto step 5.
5.  if  the htmlsrc.Contains("make")||
htmlsrc.Contains("brand") &&
    htmlsrc.Contains("model")||htmlsrc.Contains("frame
work")||htmlsrc.Contains("example")
    ||htmlsrc.Contains("models")  &&
htmlsrc.Contains("city") || htmlsrc.Contains("state") ||
    htmlsrc.Contains("cities") &&
htmlsrc.Contains("search") && htmlsrc.Contains("price")
||  htmlsrc.Contains("cost") ||
htmlsrc.Contains("budget") ||  htmlsrc.Contains("range")
&&  htmlsrc.Contains("car"))|| htmlsrc.Contains("used
car") || htmlsrc.Contains("submit")
    ||htmlsrc.Contains("search")|| htmlsrc("go");
    then  Goto step 6. Else Goto step 1.
6.  Make a list  of URLs i.e the list of web search
interfaces
```

Fig 3: Modified Algorithm for Interface detection and extraction

3.2.3 Unification of Interfaces:

Structured websites tend to publish different product features and properties. For example product car has following properties (make, model, city, state, price). Website w1 want to give (make, model, city, price) fields in the search form, website w2 may select make, model, state, city only. There would not be problem with user, but problem is for someone who collects and compiles this information. There is no proper solution for this. We can take the union of two attribute list like (make, model, city, price) ⋃ (make, model, state, city)=(make, model, state, city, price). The prototype for unification is shown in figure 4 .

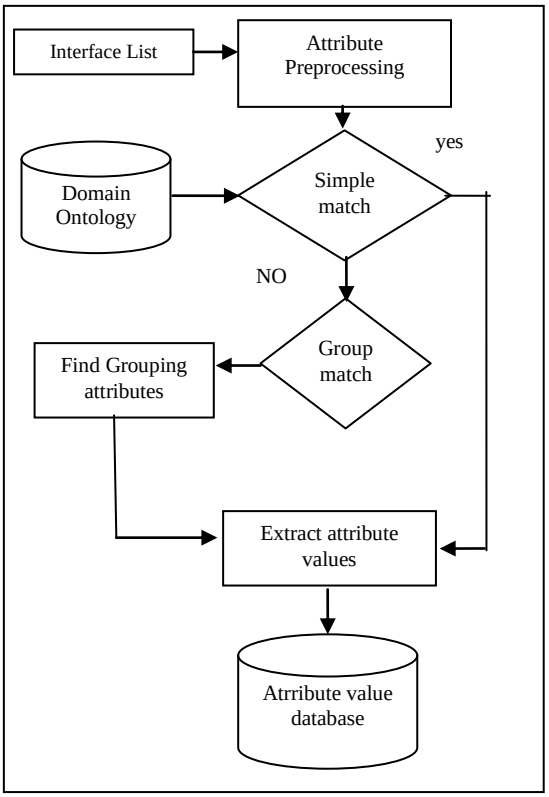

Fig 4 : Framework for Interface Unification

This model works by taking the interfaces from interface repository. Once the interface is selected, its attributes are sent for preprocessing. After this phase, each attribute is matched for simple match and group match. If it is simple match then attribute name remains same as in the ontology table. If there is group match then grouping attributes are noted down for further use.

3.2.3.1 Attribute Preprocessing:

After the extraction, attributes are preprocessed for stop words removal and stemming.

- Stop words removal are the words which are filtered out prior to, or after, processing of text. It is controlled by humans. The frequent occurring words like or, and, of, all, from, for, the, select are stop words that are removed from the attributes because they have a little importance while matching with domain ontology. Some symbols like underscore, hyphen are also removed.
- Word stemming means taking the stem of a word and generating common variants of the word. As an example, if the search text is throws then the word stem is throw and common variants of this stem include thrower, throwers and throwing. FindinSite uses rules to check to see if the word stem or any of the variants exist in the search database. If they do, then these words are added as alternatives to the original text. When a word is stemmed the queries "sale" and "sales" will return the same results.

Ontology table shown in table 2 acts as matching array which stores the attributes in Alist[i][0] and their synonyms are stored in corresponding columns. To compare, the attributes from the interfaces(from interface repository) are stored in 2-dimensional array Nlist1 in which first column contains the n attributes and second column corresponding to that attribute contains status of matching i.e whether it is simple match or it is group match. Algorithm for storage of attributes in new list is shown in figure 5.

```
Algorithm store_attribute(Nlist)

1. For each attribute from the new interface
Nlist( b1,b2…bk)
2. l=0; s=0;
3. for ( l=0;l<k;l++)
            {   for(s=0;s<k;s++)
              {  Nlist1[l][0]=bi;   }          }
4. for ( l=0;l<k;l++)
            {  for(s=0;s<k;s++)
             {    Nlist1[l][1]=NULL;
             }          }
```

Fig5: Attribute storage Algorithm

Now the interface attributes (Nlist1) is sent for matching. If the attributes from selected interface lies inside the ontology table then it is a simple match otherwise attributes are compared for group match.

3.2.3.2 Attribute Matching:

After preprocessing phase, each interface is sent for matching. There are generally two types of matching, simple matching and group matching.

Simple match-There is simple match if attribute name occurs inside the ontology table (it is exactly same or it is synonym of that attribute. For example: there is a simple match between brand and make because it lies inside the ontology table. Algorithm for simple match is shown in figure 6.

```
Algorithm simplematch(Nlist, Alist)

1. For each attribute from the new interface
Nlist( b1,b2…bk)
2. .i=0;j=0; l=0; s=1;
3. repeat step 7while (i<n && j<m)
4. if (Nlist1[l][s] = =Alist[i][j])
            {
              Nlist1[l][s]=S;
                return 1;
                exit;    }
 else       { i++;
              j++;     l++; }
5. retun 0;
```

Fig 6 :Simple Match Algorithm

Group match-There is a group match if more than attribute combine to form a single attribute. For example, there is a group match between (make-model) and (make, model). Here, two attributes combines to make a single attribute. Algorithm for Group match is shown in figure 7 and an algorithm for unification of interfaces is shown in figure 8.

```
Algorithm groupmatch(Nlist, Alist)

1. i=0; j=0;l=0; s=1;
2. Repeat step 3 while(i<n || l<k)
3. if(Nlist1[l][0]= strconcat(a[i][j],a[i++][j])
            {  Nlist1[l][s]=G;
               return 2;
               exit;     }
  else
           i++;        l++;
4. return 0;
```

Fig 7: Group Match Algorithm

```
Algorithm Unif_interf (Alist, Nlist1)
1. For each attribute from the new interface
Nlist1( b1,b2…bn)
2. x = simplematch( bi, Alist);
3. if (x = = 1){ attribute Alist[i] remains same in
Alist; }
  else {x = groupmatch(bi, Alist);
                        if(x = =2) {          attribute
Alist[i] remains same in Alist; }
                      else Alist ⋃ bi;
4. End.
```

Fig 8: Unification algorithm

3.2.4 Filling Attribute-Value Database:

After extracting attributes and making the unified interface, next step is to make the attribute value repository from which the values for particular attribute would be taken and search query form of the different site will be filled and submitted. For each interface I(ai1, ai2,….ain) of W1 website where ai1 is first attribute of that web site wi and ai2 is second attribute of that site and so on, the attribute value is extracted by following algorithm given below in figure 9.

```
Algorithm Extract_value(Interface I1,I2,….In)

1. Avlist=φ and i=0;
2.  For each attribute one by one from Wi
3.  x=document.getElementsByTagName("option");
// reach upto the point where values

            reside .
4.Repeat steps 4 and 5  while i<
x.length.document.write(x[i].innerHTML);     // it
will write the inner text which is not in <, > tags.
5. Avlist =Avlist ⋃ x ;
6. end.
```

Fig 9: Algorithm for extraction of attribute values

For filling the search interfaces, values for the attributes get stored in tables. Data structures for these tables are shown in figure 10. Now, this database is ready for submission to local interfaces.

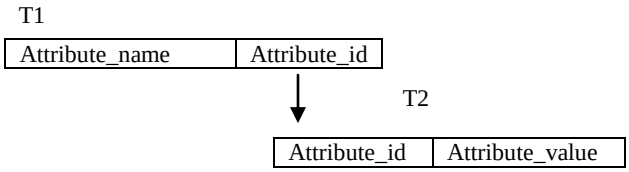

Fig 10: Attribute_value Database

This database contains one table for each attribute and one main table that take cares of the unique identifiers for that attributes. Table T1 contains all attributes under Attribute_name column and id's under Attribute id field. Table T2 contains the different attribute values for respective Attribute id under Attribute_value column.. Now, the search form is submitted using these values which will be discussed in next section.

3.2.5 Form Submission

The next phase is to fill the query forms automatically. For each attribute, the field values are extracted and unified interface is filled. Now, this interface is used to fill the local interfaces. Queries are mapped and submitted to Local interface and the result pages are extracted. This process works by selecting the interfaces one by one from interface repository. The attributes of these interfaces are compared with the attributes of unified interface built in last section. If there is simple match (means if they have same name or they are synonyms) then values for this attribute is selected from the attribute-value table, search form is filled and submitted. If there is no simple match between two attributes then attribute is sent for group match. If there is group match then two attributes from unified list that are combined to form the single attribute are selected. Attribute values for these attributes are collected from attribute-value table and concatenated. After this these values are filled in search form. There can be problems in form filling when attributes of unified interface do not match with attributes of Local Interfaces. Some of these are:

- We have attribute in Unified interface but local interface does not contain that attribute. In this case, the attribute is ignored and rest of the attribute values are filled.
- We have particular attribute in local interface but this is absent in unified interface. In this we can relax the queries.
- If we have group match then two attributes which combines to make single attribute is noted down and values of that attributes should be combined to form a single value for group attribute.

After submitting forms, result pages are returned. Now, these pages are stored in database for further processing. The analysis and integration of the returned result pages will be discussed in the next section which is the second component of the Hidden Web Search engine.

3.3 HIDDEN WEB DATA EXTRACTION, INTEGRATION AND SEARCHING

The World-Wide Web contains unstructured data as well as structured data. In case of unstructured documents, there is some structure which is enforced by the URL names and hyperlink graph. But the basic view to user is the plain text itself. On other hand, in case of structured data, the structure is enforced by relational tables[25]. Many of these tables contain both relational-style data and a small schema of labeled and typed columns, making each such table a small structured database. A large amount of Hidden web is structured i.e Hidden websites contain the information in the form of lists and tables as shown in figure 11. This dynamic information is returned to user by submitting the query in the search form.

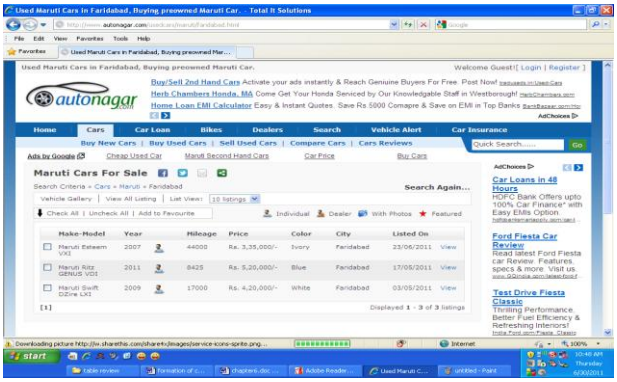

Fig 11 :Resultant page containing table data

However, visiting such sites, filling the search forms and viewing the result pages is very much time consuming task for users. Hence, it is desirable to build a prototype which will automate this process and minimize user's effort and give him high quality information in integrated form. Extracting this structured data enables one to integrate information from multiple Hidden web sources to provide value-added services, e.g., comparative shopping, meta-querying and search . Different product companies sell their products using same product features as the other companies do. Some have similar presentation style and some use different presentation as shown in figure 12.

Fig 12: Result page of carsingh.com

The basic idea is to find the information from various hidden web sources and after some processing, present it as a free web search service. This proposed component (figure 1) that will extract the table data from result page. This process works by taking result pages of a particular web site as input. The first page among all the result pages is selected. The result page contains the relevant area and irrelevant area. Relevant area is the area inside which data lies and irrelevant area contains advertisement, navigation link or another type of material that is not a part of result data. So, the next process Table Area detection and extraction detects the relevant area and discard the other ones. It selects the desired table area then this table area is sent for record area extraction.

HTML document is based on nested tags it can be interpreted as a tag tree or a DOM (Document Object Model) tree [16]. DOM is a standard for accessing and manipulating HTML documents. It presents an HTML document as a tree-structure. Using various DOM functions, we can extract the relevant area and discard the irrelevant one.

3.3.2 Table Area Detection and Extraction:

HTML is the building block of web pages. Most web pages are written in the form of HTML elements that consist of tags enclosed in angle brackets (< >). HTML tags normally come in pairs like <html> and </html>. The first tag in a pair is the *start tag*, the second tag is the *end tag* (they are also called *opening tags* and *closing tags*). In between these tags web designers can add text, tables, images, etc. Algorithm for Relevant area detection is shown below in figure 13.

```
Algo relevant_areadetect(Wi)

Wi is the ith web page from the set of webpages W.)
1. Initialize Stack as an array;
2. strsource = download(Wi);
3. Repeat step 4 to 7 for each tag in strsource until end
of file reached.
4. If the tag is a start tag  then
                    push (Stack, tag);
   If the tag is a end tag then
                    Pop top two elements A, B from
the stack
                    Where B is the Top element and A
is next to Top;
5. string Area = substring( A, B-A);
6. Analyze (Area)                    // Analyze area
for domain ontology.
7. If Area is desired area then call data_extract( Area);
   // Desired area is sent for extraction of row and
columns.
   Else continue;
```

Fig 13: Algorithm for relevant area detection

This algorithm works by selecting one by one each section from the document and analyze for the desired requirements. Input to this module is resultant webpage. An array is initialized which acts as a stack. Every tag except comment tag (!) is extracted and pushed into the stack. If the pushed tag is start tag then continue and if it is an end tag corresponding to previous stag tag then top two tags are popped from the stack and the code between tags is extracted and analyzed until the stack is empty.

Since, we are dealing with particular domain (i.e used car) therefore we can find the relevant area by looking at the domain attributes. For example, in used car domain, attributes are make, model, city, price and year. We get the result page after submitting the search. This means the result pages automatically contain domain attributes or their synonyms. To analyze every section, ontology table built in section 3.2 is used to match the fields that contain domain attributes and only the first section contains domain attributes. So, the other two sections are discarded. Now, the relevant_areadetect( ) algorithm is applied recursively into the depth of first section until it finds the real relevant area.

3.3.6 Dynamic Rule Generation:

3.3.6.1 Analyzing the behavior:

As seen in figure 11 and figure 12, the first area contains the field names under which result values are displayed. They always contain atomic values. The different behavior starts from the second row where actual values reside. The reason behind this behavior is that the information is displayed by product company takes human user into the consideration and not the extraction programs. So, number of child nodes of second area is calculated. If it is NULL then all rows are same and show same behavior .They contain data in simple table format as seen in figure 11 and it follows the data extraction process 1 and extract the data. If the second area contains any number of child nodes therefore there are again rows inside this column and all the data records follow the same pattern in each row as seen in figure 12. For this behavior, it will follow the second data extraction process and extracts data according to this process. Algorithm for detecting this behavior is given in figure 14.

```
Algo behave-detect ( )

1. Repeat step 2 for each area A2 from the set of extracted areas A;
2. y=calc_childnodes(A2);
3. S= extract first tag of A2;
4. x = document.getElementByTagName(S);
5. m = calc_childnodes(x);
6. if m = = NULL;
        Go for data extraction process with similar behavior.
   Else
        Go for data extraction process with different behavior.
7. End;
```

Fig 14: Algorithm for behavior detection

```
Algo calc.childnodes(Ai )

1. string S= extract first tag;
2. x= document.getElementByTagName(S);
3. z= x.length;
4. return z;
```

Fig 15: Algorithm for calculating children of a node

An algorithm for calculating the child nodes of node is given in figure 15. In this DOM functions are applied. This algorithm works by copying the first tag in string S. The function "document.getElementByTagename( ) returns all the childnodes of node with tag S in x. "x.length" function returns the numner of all the child nodes. This calc.childnodes( ) function returns NULL value if there is no child node linked to node.

The algorithm behave-detect( ) works by looking at this output(number of child nodes). If the node has any child node this means it shows the behavior like figure 12 and if it sees m= Null means all the columns share same behavior then it will extract the data by process1.

3.3.7 Data Extraction:

3.3.7.1 Data Extraction Process 1 and Table Formation:

One way to extract data from HTML pages is to extract all child nodes of each area. New table is created that will consists of first row as the first area where columns are filled by its child node values. The rest of the rows are maintained by rest of the areas and their columns are filled by their respective child node values. For each website, one table is created and filled according to the data packed inside its result page. If we have n websites, then we will have n tables respective to each website. The algorithm for data extraction is given in figure 16.

```
Algorithm  data_extract(Wi )

1. Repeat step 2 for each area A1,A2,…,An from the set of extracted areas A;
2. y=calc_childnodes(Ai);
3. Create table T1 with number of columns = y and number of rows = n;
4. k=0;  l=0;
5. Repeat steps 6 to 12 while (l< n)
6. j=0;
7. Repeat step 8 to 11 (while j<y)
8. S= extract first tag of Ai;
9. x = document.getElementByTagName(S);
10. T1[l][k] = x[ j ].innerHTML;
11. j++; k++;
12. l++;
```

Fig 16: Algorithm for data extraction

This algorithm works by taking all the areas as input. These areas will collectively form the table. Number of rows in this table will be number of areas and number of columns will be number of child nodes in each area. So, Each area is selected one by one and the child nodes are calculated for this area by calc_child node( ) function. First area will form the first row of table (columns of which will form the column name of table). The DOM function "document.getElementbyTagname (S) will return all the childnodes in x. The function "x[j]. innerHTML" will extract the text value of all the childnodes. These values are inserted into appropriate cells of the table.

3.3.7.2 Data Extraction Process 2 and Table Formation:

For the website that shows different behavior from the earlier one as shown in figure 137, the data should be collected and semantically labeled so that they can be appropriately organized into main repository which will be used in later searching. For example, "Maruti Suzuki WagonR Lxi BS III should be inserted into make-model column of main repository and Rs. 2,55,000 should be inserted into price column. Some sites do not provide such labels (price, make-model) when data units are encoded in the returned result page. Like in figure 12, human users can easily tell that the first line indicates make-model and second line tells the price. But the first data extraction system can not predict the labels for the data records. So, meaningful labels should be assigned to these data units. For labeling data , all the data units are identified. The value at the leaf of a tree represents the value in the result page. So, to extract the data unit, the text node values at leaf nodes of this subtree are collected. Now, these data units are sent to label assignment phase. There are some heuristics used to label the data units. These are:

***Data Type*:** Each data unit has its own semantic type. Three basic data types are considered in this approach: Date, Currency and a text string. Each type except text string has certain pattern that can be easily identified. Text string is a normal string. In general, the data units of the same concept have the same data type. For example, price is represented by integer with Rs as prefix. So, the integer values that have prefix come under the field price.

Attribute-value Type: Every search interface has some pre-defined values under each attribute. For example, make of car has a list of values from which user selects a value and submits the form. So, the data units are matched with the values in the Attribute-Value table constructed earlier.

***Prefix data matching*:** A piece of data is sometimes encoded with its label to form a single unit contains both the label and the value but without any obvious separator between them. Such nodes may occur in all rows of the result page. After data alignment, all such nodes would be aligned in one group. For example, in Figure 12, label phone contains the phone number. The line containing prefix phone followed by integer values is given below.

<tr><td>Phone : 0124-4504916   Mobile : 9711990583</td></tr>

If the same prefix lies in every row then all the data units are removed from the rows and inserted into the table under phone field.

At the end of this phase, all labels are assigned to the unlabeled data units. Now, at same position of the next row which means at the same position of the child node, data extracted and inserted under the respective field. Now, these new labels are inserted as the column names in the table and values under each label will be stored as row value under the column name.

3.3.8 Repository Formation:

After extracting rows and columns, separate tables are created for each web site that contains the result records. These tables at the end will be merged into the repository. This repository is used to find the result data corresponding to user's query.

4. EXPERIMENTS AND RESULTS:

For searching the search interfaces, test run is carried on 50 websites. The algorithm interface_detect ( ) is applied to all websites. Results are shown in figure 17 where valid means these pages are the interested pages and invalid means pages are not of our interest.

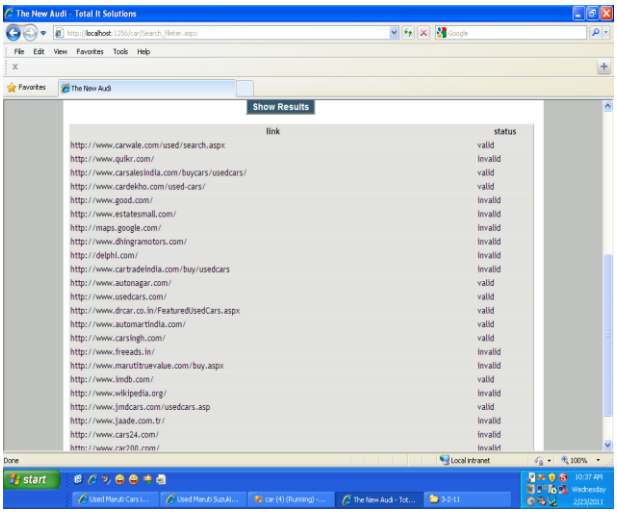

Fig 17: Results of algorithm interface_detect( )

There two metrics precision and recall are used for evaluating the correctness or accuracy of algorithm. Efficiency of algorithm interface_detect( ) is shown in table 3.

Table 3: Results for Algorithms(Interf_detect)

Algo	tn	tp	fn	fp	Precision	Recall	Accuracy
Interf_detect	12	10	0	0	100%	100%	100%

Where tn is true negative, tp is True Positive, fn is False Negative, fp is False Positive.

To implement the second part(i.e Hidden Web data extraction), test run was carried on two web sites with different behavior in car domain. The figure 18 shows the result page of www.autonagar.com and figure 19 shows the result page of www.carsingh.com.

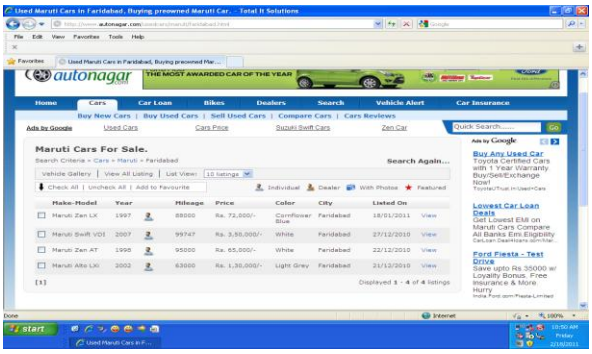

Fig 18: Result page of www.autonagar.com

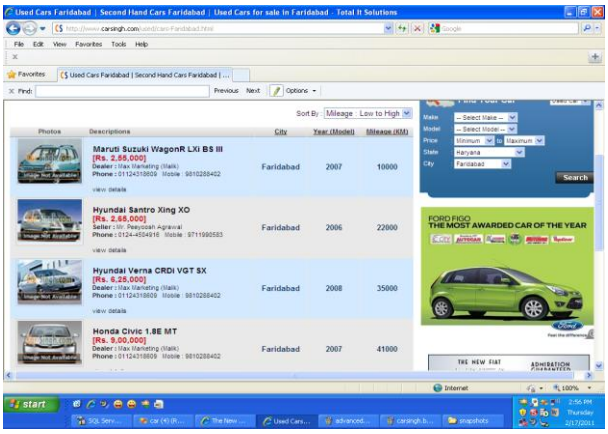

Fig 19: Result page of www.carsingh.com

These two sites are fed as the input to data extraction system . The data is extracted form the tables of two websites and stored in repository. Whe user fires the query, he/she gets the result in integrated form as shown in figure 20.

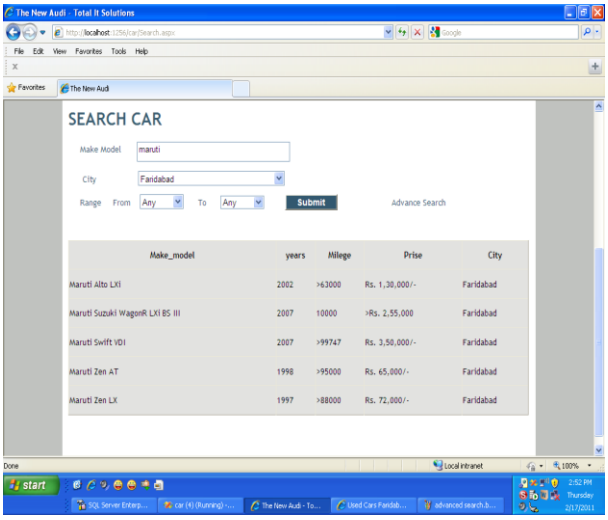

Fig 20: Result page of Hidden Web data extraction system

5. CONCLUSION

In this paper, fully automatic and domain dependent prototype system is proposed that extracts and integrates the data lying behind the search forms by searching domain features in the source code of webpage. This system searches the search interfaces in very efficient manner and also extracts the contents lying inside the table using DOM features of HTML web page and stores them at one location. It also provides user with graphical user interface where user can fill the query and can find the desired, non duplicate results in integrated form. Hence, this system reduces the user's effort in finding the desired results from Hidden web databases. In future, this work can be extended to search the user query by keyword searching approach.

6. REFERENCES

[1] Anuradha, A.K.Sharma, "A Novel Approach for Automatic Detection and Unification of Web Search Query Interfaces using Domain Ontology" selected in International Journal of Information Technology and knowledge management(IJITKM), August 2009.

[2] BrightPlanet Corp. "The deep web: surfacing hidden value."

[3] D. Florescu, A.Y. Levy, and A.O. Mendelzon. "Database techniques for the world-wide web: a survey," SIGMOD Record 27(3), 59-74, 1998.

[4] J. Wang and F. Lochovsky. "Wrapper Induction based on Nested Pattern Discovery," Technical Report HKUST- CS-27-02, Dept. of Computer Science, Hong Kong U. of Science & Technology, 2002.

[5] C.H. Chang, and S.C. Lui. "IEPAD: information extraction based on pattern discovery," Proc. 10th World Wide Web Conf. 681-688, 2001.

[6] S. Raghavan and H. Garcia-Molina. Crawling the Hidden Web. In Proceedings of VLDB, pages 129–138, 2001. SIGMOD, pages 217–228, 2003.

[7] S. Lawrence and C. L. Giles. Searching the World Wide Web. Science, 280(5360):98–100, 1998.

[8] L. Barbosa and J. Freire. Siphoning hidden-web data through keyword-based interfaces. In SBBD, 2004.

[9] Ntoulas, A., Zerfos, P., Cho, J. Downloading Textual Hidden Web Content Through Keyword Queries. In Proceedings of the 5th ACM/IEEE Joint Conference on Digital Libraries (JCDL05). 2005.

[10] Brin, Sergey and Page Lawrence. The anatomy of a large-scale hypertextual Web search engine. Computer Networks and ISDN Systems, April 1998

[11] [11] YalinWang and Jianying Hu. A machine learning based approach for table detection on the web. In WWW '02: Proceedings of the 11th international conference on World Wide Web, pages 242–250, New York, NY, USA, 2002. ACM Press

[12] H. He, W. Meng, C. Yu, and Z. Wu. WISE- integrator: An automatic integrator of Web search interfaces for e-commerce. In VLDB, 2003.

[13] Masayu Leylia Khodra1*, Dwi Hendratmo Widyantoro, "An Efficient and Effective Algorithm for Hierarchical classificationof Search Results ", Proceedings of the International Conference on Electrical Engineering and Informatics Institute Teknologi Bandung, Indonesia June 17-19, 2007

[14] S. Raghavan and H. Garcia-Molina. Crawling the Hidden Web. In Proceedings of VLDB, pages 129–138, 2001.

[15] B. He and K. C.-C. Chang. Statistical schema matching across web query interfaces. In Proceedings of SIGMOD, pages 217–228, 2003.

[16] Document Object Model (DOM), htttp://www.w3.org/DOM/

[17] B. Liu, R. Grossman, and Y. Zhai, "Mining Data Records in Web Pages," Proc. Int'l Conf. Knowledge Discovery in Databases and Data Mining (KDD), 2003, pp. 601-606.

[18] A. H. F. Laender, B. A. Ribeiro-Neto, A. S. da Silva, and J. S. Teixeira, "A Brief Survey of Web Data Extraction Tools," SIGMOD Record, 2002, vol. 31, no. 2, pp. 84-93.

[19] B. Liu, and Y. Zhai, "NET—A System for Extracting Web Data from Flat and Nested Data Records," Proc. Sixth Int'l Conf. Web Information Systems Eng., 2005, pp. 487-495.

[20] C. H. Chang, and M. R. Gigis, "A Survey of Web Information Extraction Systems", IEEE Transaction on Knowledge and Data Engineering, 2006, Vol.18, No. 10, pp. 1411-1428.

[21] N. Ashish and C. Knoblock. Wrapper generation for semi-structured internet sources. SIGMOD Record, 26(4):8{15, December 1997

[22] WordNet Search, available at http://wordnetweb.princeton.edu/perl/webwn

[23] Gruber, T. R. (1993). Towards principles for the design of ontologies used for knowledge sharing. In Guarino, N. and Poli, R., editors, Formal Ontology in Conceptual Analysis and Knowledge Representation, Deventer, The Netherlands. Kluwer AcademicPublishers.

[24] Cope, J., Craswell, N., and Hawking, D. (2003). Automated Discovery of Search Interfaces on the web. In Proceedings of the Fourteenth Australasian Database Conference (ADC2003), Adelaide, Australi,a.

[25] Simon, K., Lausen, G., and Boley, H. 2006. From HTML documents to web tables and rules. In ICEC, M. S. Fox and B. Spencer, Eds. ACM International Conference Proceeding Series, vol. 156. ACM, 125–131.

[26] Chang, K. C.-C., He, B., Li, C., Patel, M., and Zhang, Z. 2004. Structured databases on the web: observations and implications. SIGMOD Rec. 33, 3, 61–70.

[27] L. Gravano, H. Garcia-Molina, A. Tomasic,

[28] "GlOSS: Text-Source Discovery over Internet", TODS 24(2), 1999.