Chapter 5

COMPONENT DETAIL OF HWCBSE:

Search query interface forms act as entry points to the Hidden web databases containing substantial amount of high quality structured data. One has to fill these search forms manually to search the hidden web data. So, to explore the Hidden Web, its entrance must be explored first by the way of finding a search form with the help of existing directories of hidden web resources. But according to [113] even the largest search directory covers less than one sixth of the total number of web databases and has extremely low coverage for online databases. In fact, completeplanet.com [7] which is a largest of such directories, with around 70,000 databases covered only 15.6% of the total 450,000 web databases [6]. Therefore, there is a need to develop a complete directory of search interfaces available on the Web. The scale of the Hidden web suggests that such directory cannot be built by manual efforts [6]. Thus, there is a need to design a mechanism for automatic finding search interfaces.

The traditional search engines use inverted index as a data structure to index the web data and keyword interface to retrieve the data. But, surfacing the Hidden Web is more difficult task in many respects. First, the index structures for the hidden web deal with a large volume of structured data. Second, the Search query interfaces often have more than one attribute and require their respective values to be submitted. Moreover, for data sources with multi-attribute interfaces, multiple attributes and their respective values needs to be stored and the resulting HTML pages should be added into the Hidden web search engine index.

For instance, suppose a user wants to search some information (say price of a used car) hidden behind a search interface on a site. The user needs to follow the following steps.

1. He/She has to discover the URLs of the used car sites.
2. Visits homepages of all this kind of web sites.
3. Send queries through HTML forms.
4. Extract the relevant information from the result web pages.
5. Compare or integrate the results from multiple sources.

Two examples of search query interfaces are shown in figure 5.1.

User has to fill the fields of these types of forms and thereafter he/she gets the result pages. Therefore, there arises the need for new information services that can help users to find the information in integrated form. To minimize user’s effort, a “Hidden Web Search Engine” has been designed that automatically crawls, stores and searches the hidden web pages from different Hidden web sources. The following sections describe various components of the proposed search engine in detail.

5.1 HIDDEN WEB CRAWLER:

As Hidden Web is a huge repository of all domains, for testing purpose used car domain has been used. The whole process of crawling is divided into two phases. These are:

1. Learning process
2. Search interface detection and form submission

5.1.1 Learning Process:

Sampling Process - Learning process starts by collecting the web pages for particular domain. Since this work is based on used car domain, “used car” keyword is pushed into the search box of Google and result index is extracted as shown in figure 5.2. Now, out of this
large set of results, sampling process has randomly chosen four websites. These sites are downloaded to extract the search interfaces.

**Fig 5.2: Google results for used cars**

**Interface Detection and Extraction** – Next step is to identify whether these collected pages contain interface forms or not. Since a standard HTML Web form consists of form tags i.e. start form tag `<form>` and an end form tag `</form>`, crawler looks the webpage for these tags. If they are present, this means source code between these two tags is a code for search form. But this is not necessary that a given web form is also a search form, on contrary it could be a registration form. To detect and extract search query interfaces, an algorithm given below in figure 5.3 has been proposed which looks for the domain word (used car) in the URL of the web page, if it is present then it is selected and sent to the next step. Now, if URL contains strings like registration, login and sign up then this means this is a sign up form and the form is rejected.
Algorithm Interf_detect1(URL list)

Step1. Pick one by one URL from URL list
2. Check for string (domain word e.g used car) in URL. If (present) Goto step 3 else Goto Step 1.
3. Check for the string login or registration or signup in URL
   If (present) Then “it is a login/ registration form”; discard URL and Go to step 1.
   Else Goto step 4.
4. Download the source code and save.
5. check the source code for tag <form> and </form> // Form analyses
   if (present) Goto step 6
   Else “It is a simple web page”. Goto step 1.
6. check URL and 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 7.
7. check source code if ( input-type="submit" or “search” or “advance search” or “submit query” or “find” or form action="cgi")
   then Goto step 8.
   Else Goto step 1.
8. Make a list of URLs i.e the list of web search interfaces.

Fig 5.3: Algorithm for Interface Detection and Extraction

Interface Repository - As soon as Hidden Web Crawler encounters an interface, it then stores the interface in an interface repository for further processing.

This algorithm Interf_detect1( ) returns the list of domain search interfaces. But there are some incorrect results because the domain word is checked in URL but its synonyms are not considered either in URL or in its source code. For example, “auto” word can be used for “car” and there are some cases where there is no domain word in URL but they contain the search interfaces of that domain like imdb.com. To remove the limitations of algorithm given in figure 5.3, domain ontology has been built that takes attributes and their synonyms into consideration.
**Building Ontology** - To build ontology, attributes of the search forms should be extracted first. As a search form contains form fields or attributes between selection tags (i.e. `<select>` tag opens a selection list and the `</select>` tag closes it as shown in figure 5.4), crawler takes each interface from interface repository as an input to attribute extraction process and extracts the attributes using these tags. Algorithm for attribute extraction is given in figure 5.5.

```
<select name="makeone" id="makeone">option value="0">Select Make</option>
<select name="modelone" id="modelone" disabled="disabled">
<option value="0">Select Model</option>
</select>
```

**Fig 5.4: Source code showing attributes**

Algorithm **attr_extract** (URL List of interfaces)

```
Step1. Alist=φ and i=0;
  2. Pick URL one by one from list of Interfaces.
  3. x = document.getElementsByTagName("select");
      // reach upto the point where select is present.
  4. Repeat steps 5 and 6 while (i< x.length)
  5. document.write(x[i].innerHTML);
      // x.innerHTML returns the inner text which is not in <, > tags.
  6. Alist =Alist ∪ x ;
  7. end;
```

**Fig 5.5: Algorithm for Attribute Extraction**

The function attr_extract( ) takes each URL from the URL list of interfaces, and counts the number of elements proceeding with `<select>` tags. It then extracts attribute names by x.innerHTML property of HTML DOM (see Appendix A) for that count. The output of this phase is attribute list corresponding to particular site as shown in table 5.1.
Table 5.1: Extracted Attribute list

<table>
<thead>
<tr>
<th>Website</th>
<th>Attribute list</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.autonagar.com">www.autonagar.com</a></td>
<td>make, model, city</td>
</tr>
<tr>
<td><a href="http://www.carwale.com">www.carwale.com</a></td>
<td>make, model, city, price</td>
</tr>
<tr>
<td><a href="http://www.gaadi.com">www.gaadi.com</a></td>
<td>Make, model, budget, city</td>
</tr>
<tr>
<td><a href="http://www.cartradeindia.com">www.cartradeindia.com</a></td>
<td>Brand, model, state, city</td>
</tr>
</tbody>
</table>

After extracting attributes, their synonyms are extracted from wordnet (online dictionary). WordNet is a lexical database for the English language [115] that groups English words into sets of synonyms called synsets. Synonyms for all attributes are extracted from wordnet and a list is made for each attribute. Synonyms for make is shown in figure 5.6.

Fig 5.6: Synonym list for make from wordnet

Similarly, synonyms for all domain attributes are extracted and an ontology table is maintained as shown in Table 5.2.

Table 5.2: Ontology table

<table>
<thead>
<tr>
<th>Attribute(ai)</th>
<th>Synlist</th>
</tr>
</thead>
<tbody>
<tr>
<td>Make</td>
<td>Select make, brand</td>
</tr>
<tr>
<td>Model</td>
<td>Model, framework, example, poser</td>
</tr>
<tr>
<td>City</td>
<td>City, cities, metropolis</td>
</tr>
<tr>
<td>Price</td>
<td>Price, range, budget</td>
</tr>
</tbody>
</table>

To remove the limitations of algorithm interf_detect1(), modified version of algorithm has been proposed in figure 5.7 that checks domain word, its synonyms, attributes and their
synonyms in source code of the webpage. If they are present then these interfaces are extracted finally.

Algorithm Interf_detect2(URL List of interfaces)

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 (strings from synlist of Ontology Table) ||
   htmlsrc.Contains("used car") || htmlsrc.Contains("auto") || htmlsrc("secondhand 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 5.7: Modified Algorithm for Interface Detection and Extraction

5.1.2 Efficiency of Proposed Algorithms:

To evaluate the correctness or accuracy of algorithms, two widely used metrics Precision and recall have been used. Precision is defined as fraction of retrieved documents that are relevant to the search.

\[
\text{Precision} = \frac{tp}{tp + fp}
\]  

(5.1)

Recall is defined as fraction of documents that are relevant to the query that are successfully retrieved.

\[
\text{Recall} = \frac{tp}{tp + fn}
\]  

(5.2)
Where $tn$ is the number of cases that were negative and predicted negative, $tp$ is the number of cases that were positive and predicted positive, $fn$ is the number of cases that were positive but predicted negative and $fp$ are the cases that were negative but predicted positive.

Accuracy is defined as the degree of closeness of measurements of a quantity to that quantity's actual (true) value.

$$\text{Accuracy} = \frac{tp + m}{tp + m + fp + fn}$$  \hspace{1cm} (5.3)

To find the accuracy, algorithms Interf\_detect1 and Interf\_detect 2 were applied on a total of 22 number of websites and $tp$, $tn$, $fp$, $fn$ were calculated as shown in table 5.3.

<table>
<thead>
<tr>
<th>Algo</th>
<th>$tn$</th>
<th>$tp$</th>
<th>$fn$</th>
<th>$fp$</th>
<th>Precision</th>
<th>Recall</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interf_detect1</td>
<td>9</td>
<td>10</td>
<td>1</td>
<td>2</td>
<td>83%</td>
<td>90%</td>
<td>86</td>
</tr>
<tr>
<td>Interf_detect 2</td>
<td>12</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
</tbody>
</table>

**Building Attribute Value Pairs** - After extracting forms and attributes, next step is to extracts attribute values. Algorithm for attribute value extraction is given in figure 5.8.

Algorithm Extract\_value(Interface I1,I2,…,In)

Step1. $Avlist=\phi$ and $i=0$;

2. For each attribute one by one from $Wi$

3. $x=document.getElementsByTagName("option");$

   // reach up to the point where values reside.

4. Repeat steps 4 and 5 while $i< x.length$.document.write($x[i].innerHTML$);

5. $Avlist = Avlist \cup x$;

6. end.

**Fig 5.8: Algorithm for Attribute Extraction**

Attribute value extraction process takes each interface one by one from interface repository as an input and extracts the attribute values by extracting the elements form the code that are
followed by `<option>` tags. The function Extract_value() first counts the number of elements followed by `<option>` tags and then extract the elements by HTML DOM property for the count. The source code for the website www.autonagar.com shown in figure 5.9 clearly describes the values for attribute “make”.

```html
<label>Select Make</label>
<select id="make" name="make">
  <option value="">All</option>
  <option value="Audi">Audi</option>
  <option value="BMW">BMW</option>
  <option value="Chevrolet">Chevrolet</option>
  <option value="Daewoo">Daewoo</option>
  <option value="Volkswagen">Volkswagen</option>
  <option value="Willys">Willys</option>
</select>
```

**Fig 5.9: Source code showing attribute values**

Data structure for attribute value database is shown in figure 5.10.

<table>
<thead>
<tr>
<th>Attribute_name</th>
<th>Attribute_id</th>
<th>T1</th>
<th>Ti(i=1,2,3…n)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Attribute_id</td>
</tr>
</tbody>
</table>

**Fig 5.10: Attribute_value Database**

This database contains one table for each attribute and one main table that takes care 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 database has been prepared for later submissions. The next section describes form submission process in detail.
5.1.3 Search Interface Detection and Form Submission:

After learning the crawling process of particular domain, Hidden Web Crawler selects the remaining websites one by one from predetermined list of websites. It then detects the webpage for a search interface of particular domain by applying the function interf_detect2() on the source code of the page. If the webpage is recognized as a search interface, it first extracts the form and then extracts the attribute list by applying the function attri_extract().

**Attribute Matching** - As soon as these attributes are extracted, they are sent for attribute matching with the attributes of Attribute-Value table as described by algorithm Attribute_matching() given in figure 5.11. Crawler calls simple_match() function (given in figure 5.12), if there is simple match (i.e if they have same name or they are synonyms) then values corresponding to that attribute are extracted from the attribute-value table whereas if there is no simple match between two attributes then crawler calls the second function groupmatch() (given in figure 5.13). If there is group match then it finds the grouping attributes from attribute-value table that forms a single attribute and attribute values for these grouping attributes are then collected from attribute-value table.

Algorithm Attribute_matching (Alist, Nlist1)

1. For each attribute from the new interface Nlist1(b1,b2…bn)
2. x = simplematch( bi, Alist);
3. if (x = = 1)
   There is a simple match;
   else    
   x = groupmatch(bi, Alist);
   if(x = =2)
   { There is a group match; }
4. End.

**Fig 5.11: Attribute matching algorithm**

This process of matching starts with preprocessing of attributes for stopwords and stemming word removal.
• **Stop words removal** are the words which are filtered out prior to, or after, processing of text. 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. For example, if the search text is “throws” then the equivalent word stem is “throw” and common variants of this stem include words like “thrower”, “throwers” and “throwing”. FindinSite.com uses rules to check if the word stem or any of the variants exist in the search database. If they do, then there is a match.

Ontology table shown in table 5.2 acts as matching array which stores the attributes in Alist[i][0] and their synonyms are stored in corresponding columns. To compare to attributes, the attributes of the interfaces(from remaining websites) are stored in an array ( Nlist []). 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.

**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: if the attribute is brand, there exists a simple match between brand and make if it lies inside the ontology table. Algorithm for simple match is shown in figure 5.12.

```
Algorithm simplematch( )

Step1. For each attribute from the new interface Nlist( b1,b2…bk)
1. i=0;j=0; l=0; s=0;
2. Repeat step 7while (i<n && j<m)
3. if (Nlist1[l][s] = =Alist[i][j])
4. {       return 1;     exit;  }
5. else  { i++;     j++;  l++; }
6. return 0;

Fig 5.12: Simple Match Algorithm
```

75
Group match - There is a group match if more than one 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 5.13.

Algorithm groupmatch( )
___________________________________________________ ___________________________
Step1. 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])
   { return ( a[i][j] and a[i++][j]))
      // These are the grouping attributes.
      return 2;
      exit;
   }
else
   i++;
   l++;
4. return 0;
___________________________________________________ ___________________________

Fig 5.13: Group Match Algorithm

Form Submission – The next step is to submit the form. If there is simple match then crawler submits the form with values corresponding to attribute from the attribute-value table whereas if there is a group match, it finds the grouping attributes from attribute-value table that forms a single attribute and attribute values for these grouping attributes are then collected from attribute-value table. Now, search form is filled by concatenating the values corresponding to grouping attributes. The complete form submission process is shown in figure 5.14.
**Result Extraction** - When Crawler submits the form, results pages are returned in response. As every query submission results into unique URL, these URLs are now stored in index database of search engine. Since, traditional search engine maintains the index database using keywords of the page, same approach is followed in this work.

### 5.2 TRADITIONAL APPROACH:

In this approach, *Hidden Web Crawler* collects the web pages from Hidden websites. These downloaded pages are stored in repository. Repository sends necessary information to index database for searching. The challenge in this approach is - if the result page is stored like traditional search engine then structure of the Hidden web will be sacrificed. For example, if user fills the search box of search engine with query ”used Honda city 2006” and “maruti swift 2006” is also stored in index database then it will return both the results when user is
not interested in later one. So, there should be more efficient indexing style that will return accurate results. This scenario leads to the conclusion that the Hidden Web data cannot be indexed by traditional approach. So, a novel method has been proposed that considers database approach.

5.3 DATABASE APPROACH:

This approach works by retrieving the result pages after submitting the queries to search interfaces. These pages are stored in repository and again the repository sends necessary information to index database for searching. On the other side, when user gives the keyword in search box of the search engine, search engine responds with the unified search interface form for specific domain which contains attributes and will be filled by the user. If it is partially filled, then it will be filled with all the permitted values. SQL query is made automatically using attribute-value pairs in unified interface by query generator. This query is fired on index database and respective URLs are displayed to user. The whole procedure is described by system architecture as shown in figure 5.15.

![Database Approach and Result Extraction](image)

Repository sends information to index database that stores the information about the page like DT(domain type), attributes(a1, a2, ..., an), URL of the downloaded page (URLid). Hierarchical representation of Index database for two different domains is shown in figure 5.16.
5.4 QUERY GENERATION:

A template is made for construction of query. When user fills the unified interface form for specific domain which contains certain attributes, SQL query is constructed automatically using attribute-value pairs by query generator. An example query is given below:

Select URLid from T1 Where a1='x1' and a2='x2' and a3='x3';

where a1, a2, ….. are the attributes and x1, x2, x3 are the values filled in the form.

5.5 CHALLENGES IN DATABASE APPROACH:

- Since, the Hidden web contains the search interfaces with multiple attributes, there can be millions of queries that can be formed using attributes and their values.
- If number of attributes is reduced then some fields get relaxed and wasteful results can be returned.
- This approach will not be appropriate for the HTML forms that use POST method because the results for all submissions will return the same URL.
- If this approach is followed, then again it will return a huge amount of result pages to the user in response to the query leading to the problem of information overkill. Suppose, our system has indexed 100 websites and it returns 90 pages that matched to user’s query. Now, after tackling so many challenges, we are again at zero state. User has to click all the links, analyze one by one and make his mind accordingly.
- Hidden websites for particular domain can have duplicate data. For example: a user wants to sell his car then he/ she will post his advertisement in various sites. So, this approach can returns duplicate records.
Therefore, to overcome the limitations of our own (above) method, a new query submission plan has been proposed in next section which will submit less number of queries to search interfaces and extracts all results from different Hidden web sources.

5.6 FORM SUBMISSION AND ANALYSIS:

The main goal of Hidden web data extraction system is to retrieve the data that is hidden behind the search query forms. There can be many ways by which query can be submitted. These are:

5.6.1 Submission with all possible queries:

Suppose the unified interface UI(a1,a2,…..an) where a1,a2,…an are the different attributes of the interface. To fill the target forms, the value for each attribute is extracted from the Attribute-value table (Avlist). For each combination i.e ai=vij where ai is the ith attribute and vij is the jth value of ith attribute. Suppose there are on average 5 attributes within the search form and for each attribute there are more than 100 values and if there are 100 such websites then there are millions of combinations of all attribute-value pairs to form the queries and so the submissions. Based upon these submissions, there will be millions of result pages to be stored and indexed. This is really a time consuming task. So, second option could be default query as discussed in next section.

5.6.2 Submission with Default Queries:

Another way to submit query is to leave the attribute fields blank and submit. For example (make= ___ ). This is same as if a user presses the submit button without filling any field value. Like other databases, Hidden web databases also return the results by matching the query in the search form with entries in their databases. Therefore, if blank query is fired then it should return all the Hidden web data. To verify this, different hidden websites were selected and default queries were submitted. The table 5.4 shows the output for this type of query.
Table 5.4: Results with default query

<table>
<thead>
<tr>
<th>Website Name</th>
<th>Number of results</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.carsalesindia.com">www.carsalesindia.com</a></td>
<td>4000</td>
<td></td>
</tr>
<tr>
<td><a href="http://www.autoindia.com">www.autoindia.com</a></td>
<td>20277</td>
<td></td>
</tr>
<tr>
<td><a href="http://www.autonagar.com">www.autonagar.com</a></td>
<td>15290</td>
<td></td>
</tr>
<tr>
<td><a href="http://www.cartradeindia.com">www.cartradeindia.com</a></td>
<td>5000</td>
<td></td>
</tr>
<tr>
<td><a href="http://www.carwale.com">www.carwale.com</a></td>
<td>No Result</td>
<td>Constraint</td>
</tr>
<tr>
<td><a href="http://www.cardekho.com">www.cardekho.com</a></td>
<td>No Result</td>
<td>Constraint</td>
</tr>
<tr>
<td><a href="http://www.gaadi.com">www.gaadi.com</a></td>
<td>No Result</td>
<td>Constraint</td>
</tr>
</tbody>
</table>

The table data shows that the first four sites return valid results upon submitting blank query. But the last three sites restrict user to fire the blank query. They alert the user to make the selection for fields resulting in blank responses. So, the default query is not the right choice.

5.6.3 Submission with field values containing constraints:

There are some sites that contain mandatory fields that should be filled before pressing the submit button. The figure 5.17, 5.18 shows this type of sites.

Fig 5.17: Site with constraint over city field(www.carwale.com)
So, to fill the search form, mandatory field is extracted from the alert message of the submission form and then this mandatory field is matched with attribute in the attribute-table. Algorithm for filling form with mandatory field is shown in figure 5.19.

Algorithm Formfilling(I, mattr, Attribute-value table)

Step1. Repeat steps2 to 4 for each interface Ii from set of interfaces I

2. x= Find the attribute that matches mattr;
3. List=Extract value for attribute x from Attribute-value table;
4. Fill the field x of Interface Ii with List;
5. Submit the form.

In this algorithm, mattr is the mandatory attribute, I is the set of interfaces of target Hidden websites. For each interface, the attribute that matches with mandatory field is selected and all the values for that field are filled and the search form is submitted. After submitting forms, result pages are returned. Now, these pages are stored in the database for further processing so that data inside these pages are extracted and collected in a large repository which can be used for future searching efforts. The following sections describe the detailed description of extraction, processing and integration of the data from returned result pages.
5.7 DATA EXTRACTOR:

The World Wide Web contains unstructured data as well as structured data. In case of unstructured documents, the basic view to user is the plain text though underneath there is some structure which is enforced by the URL names and hyperlink graph. However, in case of structured data, the structure is enforced by relational tables. These tables contain both relational-style data and a small schema of labeled rows and columns that makes each 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. This dynamic information is returned to user by submitting the query in the search form. For example, the webpage shown in figure 5.20(b) contains the table data which is returned upon filling the search form shown in figure 5.20(a).

![Fig 5.20: (a) Search form   (b) Resultant page containing table data](image)
However, visiting such sites, filling the search forms and viewing the result pages is time consuming task for users. Hence, it is desirable to build a prototype which will automate this process, minimize user’s effort and give him high quality information in integrated form. 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. Figure 5.21 shows the semi-structured presentation style of data whereas the webpage shown in figure 5.20(b) shows structured representation. Figure 5.21 shows a web page segment containing a multiple rows in second column under description field name in contrast to the web page shown in figure 5.20(b) where each data record is displayed in the single row and the entire table data is displayed in the form of rows and columns. It would be easy to extract the information from these websites if they contain same data representation. But the World Wide Web has been dominated by HTML based on a browsing which is designed for human reading using a standard Web browser, instead of information extraction by a program. It is therefore difficult to extract the information from Hidden web by a same method as that of HTML parsing method.

In this work, a novel method is being proposed that addresses the above said issues. Before discussing the proposed system there are some important annotations about the hidden web data that are discussed first.

<table>
<thead>
<tr>
<th>Photos</th>
<th>Descriptions</th>
<th>City</th>
<th>Year (Model)</th>
<th>Mileage (KM)</th>
</tr>
</thead>
</table>
| ![Maruti Suzuki WagonR LXi BS III](image) | **Maruti Suzuki WagonR LXi BS III**  
[Rs. 2,55,000]  
Dealer: Max Marketing (Malk)  
Phone: 01124319895  
Mobile: 5810262132 | Faridabad | 2007 | 10000 |
| ![Hyundai Santro Xing XO](image) | **Hyundai Santro Xing XO**  
[Rs. 2,65,000]  
Seller: Mr. Piyush Agrawal  
Phone: 0124-4504504  
Mobile: 9711865063 | Faridabad | 2006 | 22000 |

Fig 5.21: Result page of carsingh.com
5.7.1 Basic Annotations about Hidden Web data:

HTML only defines how the data is to be displayed. But the websites under same domain show some uniformities and non uniformities in displaying their data. After surveying several sites (see figure 5.20, figure 5.21), some annotations have been made for these kinds of web pages. These are as follows:

**Similarities:**

1. Result pages are generated by specific templates. These templates are filled upon matching the fields with the query which is fired by user in search interface.
2. The data items are displayed in some repeated pattern.
3. A group of data items are always presented in contiguous region of web page. This means that the data items are displayed in the form of rows and columns.
4. The first row contains the attribute names under which the results are shown.
5. All rows except the first row show the resultant data that matches the search query.
6. All rows contain same number of columns.
7. The attribute fields have different names in different sites.
8. Generally price or cost is represented by a number itself or the number followed by the string Rs.

**Dissimilarities:**

In addition to some similarities shown by presentation of websites of same domain, they also show some irregularities. They may use the different format, syntax to present similar type of information. For example, table in the web page is created by `<TABLE>` tag and rows and columns are created using `<TR>`, `<TD>` tags. But this is not the one way to create table. They can be created by using several `<SPAN>`, `<DIV>` tags. So, due to heterogeneity and lack of same structure, it is really a challenging task to extract the data from various Hidden web sources.
5.7.2 Proposed Data Extraction System:

To overcome problems of Hidden web data extraction, "Data extraction and integration" system has been proposed in this work that does not consider tags (i.e. whether it is <TABLE> tag or it is <SPAN> tag or any other). This system extracts data by generating dynamic rule based on the position of data in the result page. The architecture of the proposed system is given in figure 5.22.

This process starts by taking result pages of a particular web site as input from which every page is selected for data extraction. It may be noted that relevant area is the area inside which the data lies (table) and irrelevant area is the area that contains advertisement, navigation link or another type of material (that is not a part of result data). Since, these result pages contain the relevant area and irrelevant area, the next method "Table Area Detection and Extraction" has been developed to detect and extract the relevant area (table). The figure 5.23 shows the table area, irrelevant area and the data record areas.
After extracting appropriate table area, next step is to extract the data record areas. These areas are actually the rows of the table. The first row always contain all the field names of table and rest of the rows show the result records under their respective field names. It is seen from figure 5.20(b) and figure 5.21 that number of columns in each row is same. In each column every row shows the same behavior. So, if all the columns of one row show same behavior then this means that data is packed into simple rows and columns of a table. Since the result page shown in figure 5.20(b) is of structured kind, data records can be easily extracted by simple method “Structured data extraction”. If all the columns of one row do not show the same behavior as shown in figure 5.21, second method “Semi-structured data extraction” has been developed that analyzes the behavior of first row and generate a dynamic rule to extract data. Since all the result pages of a particular website follow the same pattern, data in these pages are also extracted by the same rule. Therefore for each website, a dynamic rule is created and results are extracted according to it. Now, the next step is to insert this data simultaneously in the table. After maintaining the table for each website, the main repository has been maintained to store all this information in a way that makes it available and searchable to user. A detail description of whole process is given in following sections.
5.7.3 Table Area Detection and Extraction:

The first step to extract the data from result page is to locate the area of the web page where information lies. To extract the main relevant area, other irrelevant area like advertisement, company information etc. has to be omitted. Since an HTML document is based on nested tags, the pages can be interpreted as a tag tree or a DOM (Document Object Model) tree [28]. To understand the behavior of HTML document, plug-in software named IE Dom Inspector is used in this research. The DOM tree for the webpage (shown in figure 5.20(b)) is shown in figure 5.24.

Figure 5.24 clearly describes that the whole document is divided into three sections. So, among all these sections, the section that contains the desired relevant area should be extracted and others should be discarded. Algorithm for Relevant area detection is shown below in figure 5.25.
Algo relevant_areadetect(Wi)

Step 1. Initialize Stack as an array;
   2. strsource = download(Wi);
      // Wi is the ith web page from the set of webpages W.
   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;
   8. End;

Fig 5.25: Algorithm for table detection

This algorithm works by selecting one by one each section from the document and analyzing it for the domain features. An array is initialized which acts as a stack and every tag except the comment tag (!) is extracted and pushed into the stack as shown in figure 5.26(b).

If the pushed tag is a start tag then continue and if it is an end tag corresponding to previous start tag then top two tags are popped from the stack and the code between tags is extracted and analyzed for domain features until the stack is empty.
5.7.3.1 Domain Attribute Matching:

Since, “used car” domain has been chosen in this research, the relevant area can be found by looking at the domain attributes like make, model, city, price and year and their synonyms and because the result page is extracted by submitting the search interface form, this means the result pages automatically contain domain attributes or their synonyms. To analyze every section, ontology table (Table 5.2) is used to match the fields that contain domain features. Figure 5.27, figure 5.28, figure 5.29 show that only the first section contains domain attribute. So, the other two sections are discarded.

Fig 5.27: Lines containing domain attributes in first section
Now, the relevant_areaDetect() algorithm (given in figure 5.25) is applied recursively into the depth of first section until it finds the real relevant area (table). The resultant area <DIV class="linewrap3"> corresponds to main table. It is returned by this recursive algorithm as shown in figure 5.30.
5.7.3.2 Representation of Data in Hierarchical form:

The nested structure of HTML tags automatically forms a tag tree. This structure reflects the parent-child relationship among the DOM Tree nodes. The DOM tree corresponding to webpage segment (fig 5.20 (b)) is shown in figure 5.31.

Fig 5.31: Tree Representation of webpage of figure 5.20(b)
By looking at the DOM tree shown in figure 5.31 and the basic annotations about hidden web sites discussed in section 5.7.1, some conclusions have been made. These are:

1. From the point 6, it is clear that all the sub trees of the DOM tree (all rows of the table) that contain the resultant data have same number of child nodes (number of columns in each row).
2. Point 4 says that the first child node will be inserted as the first row in the table and they will be displayed as attribute names.
3. Point 5 says that the rest of the child nodes will be inserted as the rows of the table under respective attribute name.

5.7.4 Extraction of Record Areas:

To extract the data from the result page, algorithm (relevant_areadetect() ) is applied recursively applied to the first section and the areas inside the first section that contain the domain ontology are extracted. Now, number of childnodes is calculated for each area. The areas with same number of child nodes are selected because from point 6, each row in the table contains same number of columns as shown in figure 5.20 and these areas are fed as input to dynamic rule generation process.

5.7.5 Dynamic Rule Generation:

5.7.5.1 Analyzing Presentation Style:

It is clear from the figure 5.20, the first area contains the field names under which result values are displayed and this area always contain atomic values. The different behavior starts from the second row where actual values reside. So, number of child nodes of second area is calculated. If it is NULL for all the columns then this means data is in structured table format as shown in figure 5.32 and “Structured data extraction method” is followed to extract the data.
Whereas if the second area contains any number of child nodes for any column then this means there are again rows inside a column and all the data records of the page follow the same pattern in each row as shown in figure 5.33.

For this presentation style, the second data extraction process “Semi-structured data extraction” method is followed to extract data. Algorithm for detecting this behavior is given in figure 5.34.

Algorithm behave-detect ( )

Step1. Set Flag=0;


3. x= S. childnodes.length;

4. Extract start tag of A2.

5. String S1= start tag.

6. For i=0 to x-1
   if (S1.childnode[i].childnodes.length== 0)
      i++;
   else Flag=1; save i;

7. If (Flag = =0)
   call structured_dataextraction( );
   else call semistructured_dataextraction(i );

8. End;

Fig 5.34: Algorithm for behavior detection
The function `behave_detect( )` starts by calculating the number of child nodes of area A1 (i.e. x) and for that number of child nodes (x), it checks whether the second area contains any nested structure. If there is a nested structure then there are multiple rows inside that particular child node and this function calls first function `semistructured_dataextraction( )` otherwise it calls the function `structured_dataextraction( )`.

### 5.7.5.2 Structured Data Extraction:

To extract data from table of an HTML page containing structured presentation of data, all child nodes (areas) are extracted from relevant area (table). New table is maintained that consists rows equal to the number of areas and columns inside each row equal to number of childnodes of each area. Algorithm of structured data extraction is shown in figure 5.35.

```
Algorithm structured_dataextraction(Wi )

Step1. Repeat step 2 for each area A1,A2,....,An from the set of extracted areas A;

2. Extract start tag of Ai.
4. y= S. childnodes.length;
5. Create 2-dim array list with number of columns = y and number of rows = n;
   // where n is maximum number of areas.
6. k=0; l=0;
7. Repeat steps 8 to 14 while (l< Number of areas)
8. S1= extract first tag of Ai;
9. x = document.getElementById(S1);
10. j=0;
11. Repeat step 12 to 13 (while j < number of columns)
12. T1[l][k] = x. Childnode[j];
13. j++; k++;
14. l++;
15. Save table T1;
16. End.
```

Fig 5.35: Algorithm for Structured data extraction

95
The behavior of the above algorithm is illustrated by an example given below.

**Example:**

To verify this algorithm, the DOM tree of webpage shown in 5.31 is taken as an example. Area A1, A2, A3, A4 are the areas that contain domain keywords and have same number of children (10).

Number of areas = 4, Number of child nodes in each area = 10

So, Table T1 is created with 10 columns and 4 rows.

N = 4 and y = 10; K=0; l=0; First tag of First area = <P>

T[0][0] = x. childnode[0] = “ “

T[0][1] = x. childnode[1] =” make-model”


T[0][4] = x. childnode[4] =“ Price“ and so on...

Similarly, all the child nodes are extracted from tree and inserted into the table. The figure 5.36 shows webpage segment, its tree representation and the insertion of first area as first row and its child nodes as columns of that row into the table.

<table>
<thead>
<tr>
<th>Make-Model</th>
<th>Year</th>
<th>Mileage</th>
<th>Price</th>
<th>Color</th>
<th>City</th>
<th>Listed On</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;P&gt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;SPAN&gt;</td>
<td>&lt;SPAN&gt;</td>
<td>&lt;SPAN&gt;</td>
<td>&lt;SPAN&gt;</td>
<td>&lt;SPAN&gt;</td>
<td>&lt;SPAN&gt;</td>
<td></td>
</tr>
<tr>
<td>Make-Model</td>
<td>Year</td>
<td>&lt;img&gt;</td>
<td>Mileage</td>
<td>Price</td>
<td>Color</td>
<td>City</td>
</tr>
<tr>
<td>&lt;SPAN&gt;</td>
<td>&lt;SPAN&gt;</td>
<td>&lt;SPAN&gt;</td>
<td>&lt;SPAN&gt;</td>
<td>&lt;SPAN&gt;</td>
<td>&lt;SPAN&gt;</td>
<td>Listed on</td>
</tr>
<tr>
<td>Make-Model</td>
<td>Year</td>
<td>&lt;img&gt;</td>
<td>Mileage</td>
<td>Price</td>
<td>Color</td>
<td>City</td>
</tr>
<tr>
<td>&lt;SPAN&gt;</td>
<td>&lt;SPAN&gt;</td>
<td>&lt;SPAN&gt;</td>
<td>&lt;SPAN&gt;</td>
<td>&lt;SPAN&gt;</td>
<td>&lt;SPAN&gt;</td>
<td>Listed on</td>
</tr>
<tr>
<td>Make-Model</td>
<td>Year</td>
<td>&lt;img&gt;</td>
<td>Mileage</td>
<td>Price</td>
<td>Color</td>
<td>City</td>
</tr>
<tr>
<td>&lt;SPAN&gt;</td>
<td>&lt;SPAN&gt;</td>
<td>&lt;SPAN&gt;</td>
<td>&lt;SPAN&gt;</td>
<td>&lt;SPAN&gt;</td>
<td>&lt;SPAN&gt;</td>
<td>Listed on</td>
</tr>
</tbody>
</table>

Fig 5.36: Data extraction process for first area
Figure 5.37 shows webpage segment, its tree representation and the insertion of second area as second row and its child nodes as columns of that row into the array list and this process repeats until all areas are extracted.

The algorithm data_extract( ) is applied recursively and extracted data is inserted in the form of rows and columns and form a table at the end as shown in table 5.5.

<table>
<thead>
<tr>
<th>Make-Model</th>
<th>Year</th>
<th>Mileage</th>
<th>Price</th>
<th>Color</th>
<th>City</th>
<th>Listed on</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maruti Esteem VXI</td>
<td>2007</td>
<td>44000</td>
<td>3,35,000</td>
<td>Ivory</td>
<td>Faridabad</td>
<td>23/06/2011</td>
</tr>
<tr>
<td>Maruti Ritz GENUS VDI</td>
<td>2011</td>
<td>8425</td>
<td>Rs 5,20,000</td>
<td>Blue</td>
<td>Faridabad</td>
<td>17/05/2011</td>
</tr>
<tr>
<td>Maruti Swift DZire LXI</td>
<td>2009</td>
<td>17000</td>
<td>4,20,000</td>
<td>White</td>
<td>Faridabad</td>
<td>03/05/2011</td>
</tr>
</tbody>
</table>

5.7.5.3 Semi-Structured Data Extraction:

For the website that contains semi-structured presentation (result page shown in figure 5.33), the data should be collected and semantically labeled so that they can be appropriately organized and stored into main repository. For example, attribute name for “Maruti Suzuki WagonR Lxi BS III is “description”. So, this data 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 (e.g., price, make-model) when data units are encoded in the returned result page. Like in figure 5.33, human users can easily tell that the first line indicates make-model and second line tells the price. But the function `structured_data_extraction()` can not predict the labels for the data records. So, meaningful labels have to be assigned to these data units. The next section describes the HTML DOM structure of webpage (shown in figure 5.21) which is used to extract the semi-structured presentation of data.

**Representation of Data in Hierarchical form:**

Tag tree representation of the web page segment (figure 5.21) is shown below in figure 5.38.

![Fig 5.38: Tree representation of webpage (figure 5.21)](image)

To label the data items, *Data preprocessing system* has been proposed (shown in figure 5.39) that preprocesses data items. This tag tree shown in figure 5.38 shows that when `behave_detect()` (shown in figure 5.34) is applied on the web page tree, second area returns the child node’s location that contains nested structure and this child node (subtree) is sent for preprocessing.
This system works in five phases. In the first phase, all the data units are extracted. 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. Some heuristics were used to label the data units. These are:

**Data Type:** Each data unit has its own semantic type. Two basic data types are considered in this approach: Date and a text string. Each type, except text string has certain pattern that can be easily identified like date can be represented by DD-MM-YY or DD/MM/YY. In general, the data units of the same concept have the same data type, therefore the data units that have any variant of date as data type come under the label “date”

**Prefix data matching:** A piece of data is sometimes encoded with its label to form a single unit containing both the label and the value without any separator between them. Such nodes may occur in all rows of the result page. For example, price is represented by integer with Rs as prefix. So, the integer values that have Rs as prefix come under the field “price” and similarly, label phone contains the prefix phone number. The line containing prefix phone followed by integer values is given below.

| <tr><td><strong>Phone : </strong>0124-4504916 &nbsp; 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.

**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 in section 5.1. Example of this is shown in figure 5.40 in which Hyundai is value of attribute make and Santro Xing is a value of attribute model. So, “Hyundai Santro Xing combined to come under the group make-model.

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 is 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 are stored as rows under the respective column name. Table 5.6 shows the new table formed after label assignment phase.

Table 5.6: Table formed after extracting data from all the areas

<table>
<thead>
<tr>
<th>Photos</th>
<th>Make-model</th>
<th>Price</th>
<th>Phone</th>
<th>City</th>
<th>Year</th>
<th>mileage</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Algorithm for Label assignment is shown in figure 5.41.
Algorithm Label-assign( )

Step1. Check t for datatype_checking;
    if yes Goto step 4
2. Check t for prefix matching ;
    if yes . Extract prefix Goto step 5
3. Check t for attribute in Attribute Value table;
    if yes Goto step 6
4. label = Date ;
5. label = prefix;
6. label = attribute;
7. return label ;
8. End;

Fig 5.41: Algorithm for Label assignment

5.8 REPOSITORY FORMATION:

After extracting data from all the hidden web sources, the next step is to integrate the data in the large repository. For each website, a table is created. But this system does not know what regular data records are useful to a user. It simply finds all of them. However, different websites of same domain presents different product features. However, in a particular application, the user is generally interested in some specific type of data records. For example, in car domain, user is generally interested in make, model, price, year, mileage, city of the car. Hence, this system has been designed to output only certain important features of a product. One main table is maintained with some selected product features as the column names and data the matches these column names from different tables of various sites will be inserted as rows under respective columns. This table will act as repository for later search. Algorithm for repository formation is given in figure 5.42.
Algorithm repository_form( )

Step 1. Repeat step 2 to 4 for each column (X) of main table MT
  2. Repeat step 3 to 4 for each column (Yk) of table Ti (Table T1, T2, … from table collected from all web sites)
  3. x= match (MT(X), T(Yk));
  4. if (x = = 1) Repeat step 5 to 8
     i++; j++;
  6. count1 = i ; count2 = j;
  7. Repeat step 8 For (i=0 to i < count1 and For j=0 to j < count2)
  8. Extract all rows for column name A[j];
  9. INSERT all rows INTO MT ( A[i] )VALUES ( all rows with column name A[j] FROM TABLE Ti );
 10. k++;
     GOTO step 2;
 11. End;

Fig 5.42: Algorithm for repository formation

Algorithm match ( MT (X), Ti (Y) )

Step 1. y= simple_match (MT(X), T(Y))
  2. if (y = = 1)
     return 1 ;
  else
     y= attribute_valuematch (MT(X), T(Y))
     if (y = = 1)
        return 1
     exit ;
  else
     return 0 ;
  3. End;

Fig 5.43: Algorithm for column matching
Algorithm simple_match (MT(X), T(Y))

Step1. var str = MT(X);
    2. var patt1 = T(Y);
    3. x = str.match(patt1);
    4. if(x == NULL);
        check whether Y is a synonym of X // check in the ontology table
        if there is match then return 1 and exit;
        else return 0;
    5. End;

Fig 5.44: Algorithm for simple matching

Algorithm attribute_valuematch(MT(X), T(Y))

Step1. Extract first five rows under column Y;
    2. Check the Attribute value table // section 5.1.2
    3. Return attribute under which these values lie.
    4. If only one attribute (L1) is returned
        Call simple_match(MT(X), L1); Else
        L3 = concat(L1, L2);
        Call simple_match(MT(X), L3);
    5. End;

Fig 5.45: Algorithm for attribute value matching

The algorithm repository_form( ) works by taking main table as first input where the column names are important features of a product and second input is the table “Ti” from each website. Each column from main table MT is compared with every column of table Ti. The function match( ) given in figure 5.43 will find whether the two columns match or not and if they match then all the rows are copied to the main table MT.

There can be different types of matching. If the columns(X, Y) have same name then it is simple matching and if they do not have same name, they can be synonyms. So, column of Ti is checked for synonyms in ontology table (built in section 5.1.1 and shown in table 5.2). If it
exists in ontology table, then function simple_match( )(given in figure 5.44) returns 1 and all rows under column Y are copied to rows under column X.

If the columns do not match then function match() calls another function attribute_valuematch( ) (given in figure 5.45) that extracts first few rows under column Y and these rows are now compared with values from attribute-value table(built in section 5.1, figure 5.10). If these values lie inside the attribute value table then that attribute is returned and this attribute is again sent for matching. Now, if they match then all rows are obviously copied to main table.

A two-dimensional array A[] has been used to store various column names of MT table and the matched column names of table Ti. Column name of MT table is stored at ith location of A and jth location stores column name of Ti. After getting information about the column matching, MT table is filled by inserting values under column name A[j] into the field name A[i]. At the end of this phase, a repository is formed which contains data from all the tables corresponding to different hidden websites. Now, this repository sends data records to index database which is discussed in next section.

5.9 INDEXER:

Traditional search engines offer keyword based search interface to search information on World Wide Web where user does not require to provide any knowledge about the processing of his/her query. It searches documents for specified keywords and returns a list of the documents where the keywords are found. In contrast to text databases that are queried by set of keywords, a relational database consists of set of records and accordingly SQL statements have to be issued to retrieve the required relational information.

To make the HWCBSE (Hidden Web Search Engine) work like traditional search engine, keyword based search interface is provided to user where the user can enter set of keywords and he/she does not have to know about the processing and schema of database. This thesis proposes an inverted index that supports keyword based search over the relational databases.
As discussed above, Hidden web databases are crawled in advance and results extracted from these databases are stored in main repository. The indexer extracts necessary information like attribute names and their respective result values from the repository and builds an *Inverted Index* as shown in table 5.7. Let’s suppose a1, a2, a3…, an are the attributes, v11,v12,….v1n are the result values under attribute a1, v21,v22,….v2n are the result values under the attribute a2 and so on. Now, an inverted index is built which contains two fields: Result _value and attribute name.

<table>
<thead>
<tr>
<th>Result_value</th>
<th>Attribute_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>v11</td>
<td>a1</td>
</tr>
<tr>
<td>v12</td>
<td>a1</td>
</tr>
<tr>
<td>V13</td>
<td>a1</td>
</tr>
<tr>
<td>v21</td>
<td>a2</td>
</tr>
<tr>
<td>v22</td>
<td>a2</td>
</tr>
<tr>
<td>v23</td>
<td>a2</td>
</tr>
<tr>
<td>v31</td>
<td>a3</td>
</tr>
<tr>
<td>v32</td>
<td>a3</td>
</tr>
<tr>
<td>v33</td>
<td>a3</td>
</tr>
</tbody>
</table>

Now, when user enters a query in search box of search interface, *Query Processor* (next component of the search engine) searches the keywords of query in the inverted index and then construct SQL query using these keywords and attributes. The detail description of *Query Processor* is given in next section.

**5.10 QUERY PROCESSOR:**

When user fires query in the search box of HWCBSE (Hidden web crawler based search engine), *Query Processor* processes this query to find relevant records. The complete framework for Query processing is shown in figure 5.46.
Algorithm for processing of user query is given in figure 5.47.

Algorithm Queryprocess(query)

1. String []keywordarraylist= split(query);
2. String query1 = " ";
3. Repeat Steps For i=0 to i< keywordlist.length
   // for all individual keywords of a query
4. A = SELECT Attribute_name FROM INDEX_DATA BASE
   WHERE Result_value = keywordlist[i];
    // where Attribute_name, Result_value are the fields of Index database
5. query1 = query1+ " A LIKE '%" + keywordlist[i] + "%' and";
6. SELECT * FROM MT WHERE + query1;
    // where MT is the main repository of data records
7. End.

Fig 5.47: Algorithm for Query processing

*Query Processor* first splits the query in keywords and now these individual keywords are searched for their attribute names in inverted index. A blank SQL query template, constructed in advance, builds a dynamic query based on these attribute names and corresponding keywords. This SQL query is now fired at main repository to obtain data
records. Now, the last phase is interaction of user to this search engine. The detail description of last phase is discussed in next section.

5.11 USER INTERACTION WITH HWCBSE:

A user-friendly search interface has been provided for users to enter their queries wherein the user enters the query in the search box. The query is processed by *Query processor* and relevant results returned thereof are provided to the user. User interaction process is given in figure 5.48.

![Fig 5.48: User Interaction](image)

The screenshot of user search interface with a search box that provides the Keyword based search to user is shown in figure 5.49.

![Fig 5.49: Search Box of HWCBSE](image)

HWCBSE has been implemented for two web sites with different behavior (structured and semi-structured) in used car domain. The results obtained thereof are shown and discussed in the next chapter.