IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661,p-ISSN: 2278-8727, Volume 17, Issue 4, Ver. IV (July – Aug. 2015), PP 75-90
www.iosrjournals.org

Integration of Search Interfaces in Deep Web Databases for Specific Domains

Deepak Kumar[1], Dhanesh kumar[2], Dr. Mahesh Kumar[3]
[1](MDU Rohtak, India) [2](MRKIET Rewari, India),[3](MRKIET Rewari, India)

Abstract : *The amount of information on Deep Web is growing rapidly, as most of the organizations put their content online behind the search forms. The information behind the search forms is often of very high quality and can be extremely valuable to many users. Therefore there has been increased interest in retrieval of deep web data. The conventional crawlers retrieve only publicly indexable web ignoring the large amount of high quality information deep behind search forms hence there is a need to build a deep web crawler that can crawl to deep web. But the main challenge for a deep web crawler is to model a query interface and to generate meaningful queries for the search interfaces. In this paper authors have integrated the search interfaces on the basis of the semantic relationships to generate the Generalized Search Interface.*

I. Introduction

The Internet is a worldwide, publicly accessible series of interconnected computer networks. Many people use the terms Internet and World Wide Web interchangeably, but in fact the two terms are not synonymous. The Internet and the Web are two separate but related things. The Internet is a collection of interconnected computer networks, linked by copper wires, fiber-optic cables, wireless connections, etc. In contrast the World Wide Web (or the "Web") is a system of interlinked, hypertext documents accessed via the Internet. The World Wide Web is one of the services accessible via the Internet, along with various others including e-mail, file sharing, online gaming etc. With a Web browser, a user views Web pages that may contain text, images, and other multimedia and navigates between them using hyperlinks. The web browser is a software application that enables a user to display and interact with text, images, and other information typically located on a Web page at a website on the World Wide Web or a local area network. Text and images on a Web page can contain hyperlinks to other Web pages at the same or different website. Web browsers allow a user to quickly and easily access information provided on many Web pages at many websites by traversing these links. Searching is one of the most used actions on the Internet. Search engines as an instrument of searching, are very popular and frequently used sites. A program that searches documents for specified keywords and returns a list of the documents where the keywords were found. Search engine usually refers to a Web search engine, which searches for information on the public Web. Typically, a search engine works by sending out a crawler to fetch as many documents as possible. Another program, called an indexer, then reads these documents and creates an index based on the words contained in each document. Each search engine uses a proprietary algorithm to create its indices such that, ideally, only meaningful results are returned for each query.

A large amount of information on the web today is available only through search interfaces; in this scenario the user have to type a set of keywords in a search form in order to access the pages from different web sites. So the traditional search engines retrieve content only from the publicly indexable web, i.e., the set of Web pages extracted purely by following hypertext links, ignoring search forms and pages that require authorization or prior registration. This portion of web is called deep web. The deep Web is qualitatively different from the surface Web. Deep Web sources store their content in searchable databases that only produce results dynamically in response to a direct request. But a direct query is a "one at a time" laborious way to search. Traditional search engines cannot "see" or retrieve content in the deep Web those pages do not exist until they are created dynamically as the result of a specific search. Because traditional search engine crawlers cannot probe beneath the surface, the deep Web has heretofore been deep. The term "deep web" was originally used to describe pages and sites that search engines failed to index, either because of the way in which the pages were designed or because the information was password protected or held in a database. This missing information is sometimes referred to as the invisible web, the deep web or the dark web. Known in research circles as the invisible, deep, or deep Web, this immersed content is an estimated 500 times larger than the surface Web, which is estimated at over four billion pages.

However, according to recent studies, the content provided by many Deep Web sites is often of very high quality and can be extremely valuable to many users. A large amount of web contents residing on deep web are generated dynamically from databases and other data sources deep from the user. These web contents are generated only when queries are asked via a search interface, rendering interface integration a critical problem in many application domains. In this paper, a novel approach for interface integration is implemented. It uses semantic relationships to integrate the search interfaces in a particular domain of interest and the tools used in implementation are Asp.Net, C#, javascript, Sql server.

II. Classification Of Text Databases

Most important thing is to define that how appropriate classification schemes for deep web databases can be achieved. And then present alternative methods for text database categorization. Some commonly used classification schemes are; -

Hierarchical Classification Scheme

It is a rooted directed tree whose nodes correspond to (topic) categories and whose edges denote specialization. An edge from category 1 to another category 2 indicates that 2 is a subcategory of 1. Alternative Classification Strategies We now turns to the central issue of how to automatically assign databases to categories in a classification scheme, assuming complete knowledge of the contents of these databases. Of course, in practice one will not have such complete knowledge, so we it have to use the probing techniques to approximate the “ideal” classification definitions.

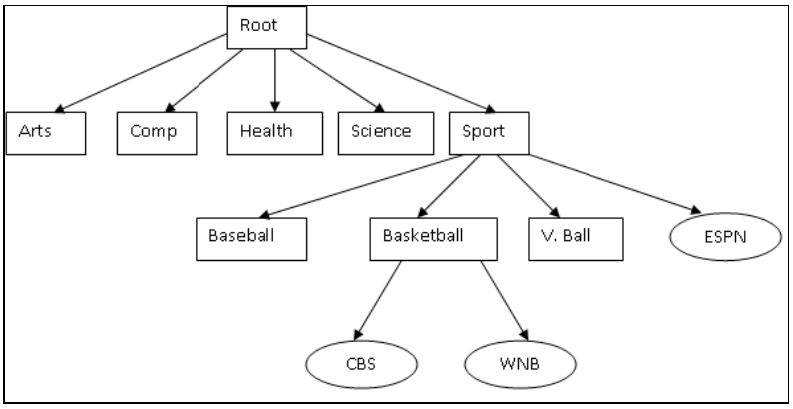


Fig. 1- Deep Web Classification Scheme

To assign a searchable Web database to a category or set of categories in a classification scheme, one possibility is to manually inspect the contents of the database and make a decision based on the results of this inspection. Incidentally, this is the way in which commercial Web directories such as Invisible Web.

Deep Web Database Model

There exists a variety of Deep Web sources that provide information on a multitude of topics. Depending on the type of information, we may categorize a Deep-Web site either as a textual database or a structured database. A textual database is a site that mainly contains plain-text documents. Since plain-text documents do not usually have well-defined structure, most textual databases provide a simple search interface where users type a list of keywords in a single search box (Fig. 2). In contrast, a structured database often contains multi-attribute relational data (e.g., a book on the Amazon Web site may have the fields title=‘Computer Organization and Architecture ’, author=‘M.M.Mano’ and ISBN=‘0899421015’) and supports multi-attribute search interfaces (Fig. 3.3).

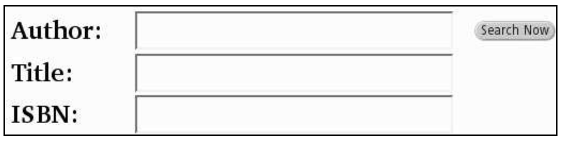


Fig. 2: A multi-attribute search interface

III. Extraction of Information from Deep Web

Crawlers are programs that automatically traverse the Web graph, retrieving pages and building a local repository of the portion of the Web that they visit. Depending on the application at hand, the pages in the repository are either used to build search indexes, or are subjected to various forms of analysis (e.g., text mining). Traditionally, crawlers have only targeted a portion of the Web called the publicly indexable Web (PIW). This refers to the set of pages reachable purely by following hypertext links, ignoring search forms and pages that require authorization or prior registration. Large portions of the Web are 'deep' behind search forms, in searchable structured and unstructured databases (called the deep Web or deep Web). Pages in the deep Web are dynamically generated in response to queries submitted via the search forms. The deep Web continues to grow, as organizations with large amounts of high-quality information (e.g., the Census Bureau, Patents and Trademarks Office, news media companies) are placing their content online, providing Web-accessible search facilities over existing databases. The problem of building a deep Web crawler is being discussed here; one that can crawl and extract content from these deep databases. Such a crawler will enable indexing, analysis, and mining of deep Web content, akin to what is currently being achieved with the PIW. In addition, the content extracted by such crawlers can be used to categorize and classify the deep databases.

A generic deep web crawling algorithm

Given that the only "entry" to the pages in a Deep-Web site is its search form, a Deep-Web crawler should follow the three steps. That is, the crawler has to generate a query, issue it to the Website, download the result index page, and follow the links to download the actual pages. In most cases, a crawler has limited time and network resources, so the crawler repeats these steps until it uses up its resources.

```
(1) While (the resources are available) do
// select a term to send to the site
(2) qi = SelectTerm ()
// send query and acquire result index page
(3) R (qi) = QueryWebSite (qi)
// download the pages of interest
(4) Download(R (qi))
(5) Done
```

The generic algorithm for a Deep-Web crawler is shown above. For simplicity, it is assumed that the Deep-Web crawler issues single-term queries only. The crawler first decides which query term it is going to use (Step (2)), issues the query, and retrieves the result index page (Step (3)). Finally, based on the links found on the result index page, it downloads the Deep Web pages from the site (Step (4)). This same process is repeated until all the available resources are used up (Step (1)). Given this algorithm, it can be seen that the most critical decision that a crawler has to make is what query to issue next. If the crawler can issue successful queries that will return many matching pages, the crawler can finish its crawling early on using minimum resources. In contrast, if the crawler issues completely irrelevant queries that do not return any matching pages, it may waste all of its resources simply issuing queries without ever retrieving actual pages. Therefore, how the crawler selects the next query can greatly affect its effectiveness.

Problem formalization

The problem of query selection can be formalized as follows: it is assumed that the crawler downloads pages from a Web site that has a set of pages S. Each Web page is represented in S as a point. Every potential query q_i that may be issued can be viewed as a subset of S, containing all the points (pages) that are returned when we issue q_i to the site. Each subset is associated with a weight that represents the cost of issuing the query. Under this formalization, the goal is to find which subsets (queries) cover the maximum number of points (Web pages) with the minimum total weight (cost). This problem is equivalent to the set covering problem in graph theory. There are two main difficulties that should be addressed in this formalization. First, in a practical situation, the crawler does not know which Web pages will be returned by which queries, so the subsets of S are not known in advance.

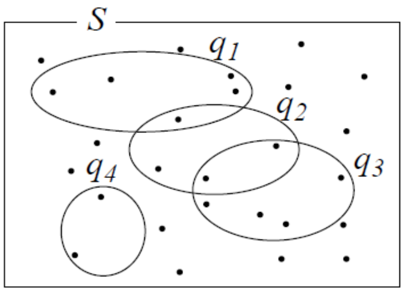


Fig 3: A set-formalization of the optimal query selection problem

Without knowing these subsets the crawler cannot decide which queries to pick to maximize the coverage. Second, the set-covering problem is known to be NP-Hard, so an efficient algorithm to solve this problem optimally in polynomial time has yet to be found. In this paper, a near optimal solution at a reasonable computational cost can be found with an approximation algorithm. The algorithm leverages the observation that although it is not known which pages will be returned by each query q_i that is issued, it can be predicted how many pages will be returned. Based on this information the query selection algorithm can then select the "best" queries that cover the content of the Web site

Deep Web Crawling

Conventional crawlers rely on the hyperlinks on the Web to discover pages, so current search engines cannot index the Deep-Web pages. We believe that an effective Deep-Web crawler can have a tremendous impact on how users search information on the Web. The only "entry" to Deep Web pages is through querying a search form, there are two core challenges to implementing an effective Deep Web crawler:
1. The crawler has to be able to understand and model a query interface, and
2. The crawler has to come up with meaningful queries to issue to the query interface.

This paper presents a solution to the first challenge, i.e. how a crawler can understand the model of different query interfaces in a particular domain and then integrate them to produce a unified query interface. Clearly, when the search forms list all possible values for a query (e.g., through a drop-down list), the solution is straightforward. The aim is to crawl the selective portion of the deep Web, extracting contents based on the requirements of a particular application or task.

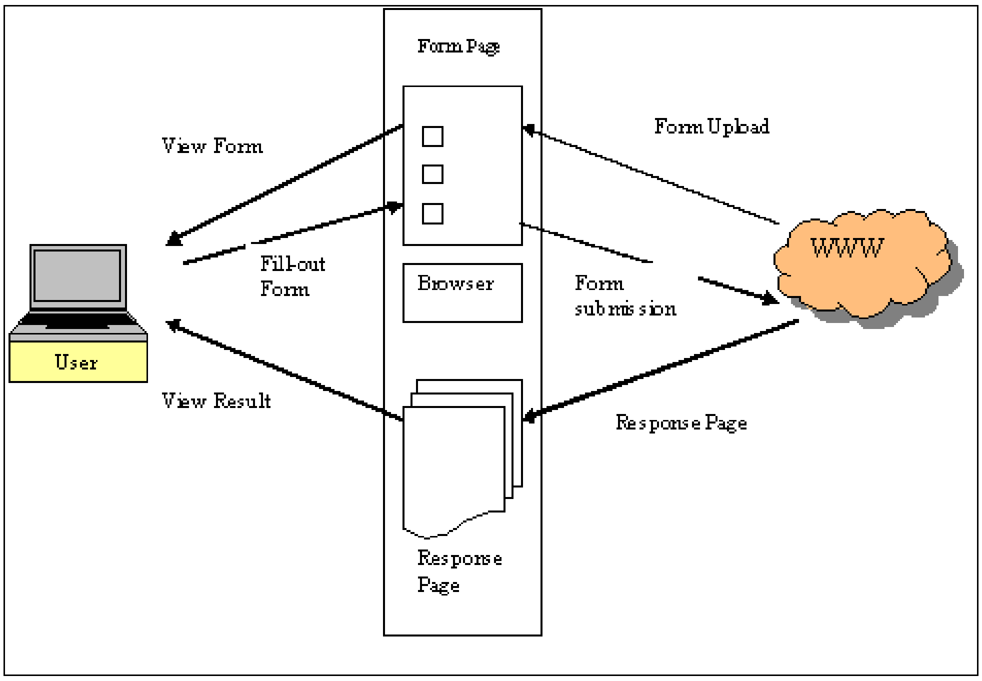


Fig 4: User form interaction

The fundamental difference between the actions of a deep Web crawler, such as HiWE, and that of a traditional crawler is with respect to pages containing search forms. Fig. 4 illustrates the sequence of steps that take place, when a user uses a search form to submit queries on a deep database. The user first get the form to be filled then the user fill-out the form and submit it to the web query front end which directly communicates with the deep database. After submission the form response page come and the user can see the resultant page. But our main aim is to fill these form by the software program on which research is going on and in order to achieve our goal we first need to unify the search interfaces. A form element can be any one of the standard input objects such as selection lists, text boxes or radio buttons. Each form element is associated with a finite or infinite domain and a text label that semantically describes the element (Fig. 5). The values used to fill out forms are maintained in a special table called the LVS (Label Value Set) table (Fig. 5). Each entry in the LVS table consists of a label and an associated fuzzy/graded set of values. (e.g., Label = ``State" and value set = {(``Haryana", 0.8), (``Punjab", 0.7)}). The weight associated with a value represents the crawler's estimate of how effective it would be, to assign that value to a form element with the corresponding label.

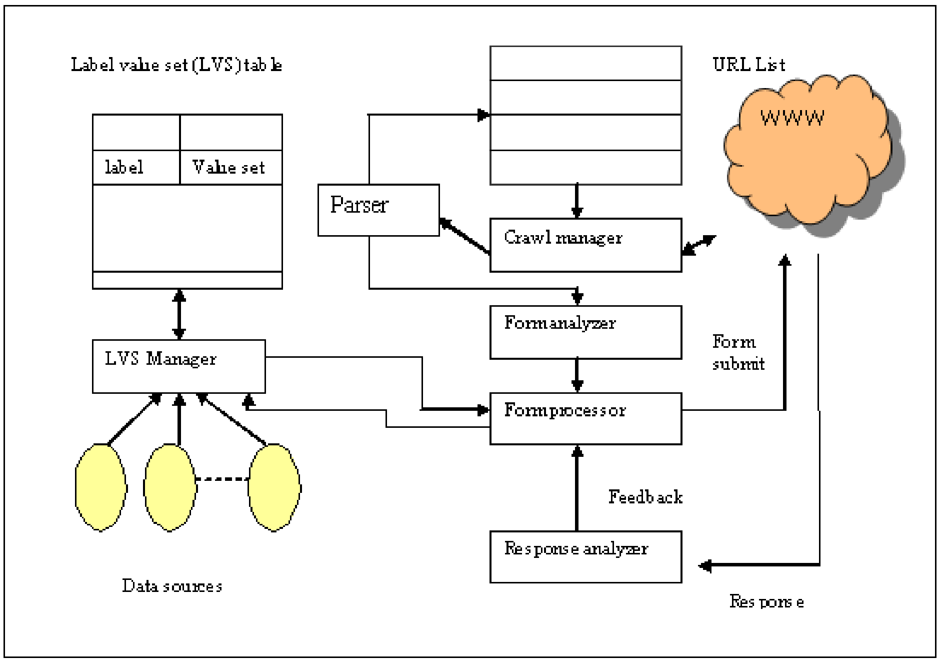


Fig 5: HiWE Architecture

Internal Form Representation

On receiving a form page, a crawler first builds an internal representation of the search form. Abstractly, the internal representation of a form F includes the following pieces of information:

$F = (\{E_1; E_2 ; : : : ; E_n\};S;M)$,

where $\{E_1; E_2 ; : : : ; E_n\}$ is a set of n form elements, S is the submission information associated with the form (e.g., submission URL, internal identifiers for each form element, etc.), and M is meta-information about the form (e.g., URL of the form page, web-site hosting the form, set of pages pointing to this form page, other text on the page besides the form, etc.). A form element can be any one of the standard input elements: selection lists, text boxes, text areas, checkboxes, or radio buttons.

Task-specific Database

In HiWE, task-specific information is organized in terms of a finite set of concepts or categories. Each concept has one or more labels and an associated set of values. For example, the label 'Company Name' could be associated with the set of values {'wipro','TCS','infosys'…….}. The concepts are organized in a table called the Label Value Set (LVS) table. Each entry (or row) in the LVS table is of the form (L, V), L is label and $V = \{v_1, v_2,……,v_n\}$ is a fuzzy/graded set of values. Fuzzy set V has an associated membership function M_v that assigns weights/grades, in the range [0, 1], to each member of the set.

Matching Function

For a form element with a finite domain, the set of possible values that can be assigned to the element is fixed, and can be exhaustively enumerated. For example, a domain Dom (E_1) has only three elements, the crawler can first retrieve all relevant articles, then all relevant press releases, and finally all relevant reports. For infinite domain elements, HiWE textually matches the labels of these elements with labels in the LVS table. For example, if a textbox element has the label "Enter state" which best matches an LVS entry with the label "State", the values associated with that LVS entry (e.g., "Haryana" or "Punjab") can be used to fill out the textbox.

1.Label Matching

There are two steps in matching form labels with LVS labels. First, all labels are normalized; this includes, among other things, conversion to a common case and standard IR-style stemming and stop-word removal. Next, an approximate string matching algorithm is used to compute minimum edit distances, taking into account not just typing errors but also word reordering (e.g., we require that two labels 'Company Type' and 'Type of Company', which become "company type" and "type company" after normalization, be identified as being very similar, separated by a very small edit distance). HiWE employs a string matching algorithm that meets these requirements.

1.Ranking value assignments

HiWE employs an aggregation function to compute a rank for each value assignment, using the weights of the individual values in the assignment. In addition, HiWE accepts, as a configurable parameter, a minimum acceptable value assignment rank, Amin. The aim is to improve submission efficiency by only using relatively 'high-quality' value assignments. Hence, to generate submissions, HiWE uses only value assignments whose rank is at least Amin. Fuzzy conjunction, average, and probabilistic functions are experimented for the ranking value assignment.

Submission Efficiency

Let N_t be the total number of forms that the crawler submits, during the course of its crawling activity. Let N_s denote the number of submissions which result in a response page containing one or more search results. Then, we define the strict submission efficiency (Ss_e) metric as:

$$Ss_e=N_s/N_t$$

This metric is 'strict', because it penalizes the crawler even for submissions which are intrinsically 'correct' but which did not yield any search results because the content in the database did not match the query parameters. Thus the submission efficiency estimate how much useful work a crawler accomplishes, in a given period of time. In particular, if two identically configured crawlers are allowed to crawl for the same amount of time, the crawler with the higher rating is expected to retrieve more 'useful' content than the other.

IV. Integration of Search Interfaces in Deep Web Databases

The integration of search interface schemas is typically accomplished in following two steps:
(1)Schema Matching identifies semantic correspondences among interface attributes;

(2)Schema integrating constructs a unified schema given the discovered mappings of the attributes. Such a unified schema should encompass all unique attributes over the given set of interfaces, and should be structurally and semantically well formed.

Query Interface Representation

A query interface specifies a conceptualization of a domain in terms of attributes and the relation between different attributes. They are typically organized into a taxonomy tree where each node presents an attribute and each attribute is a specialization of its parent. The structure of interface can be captured using a hierarchical schema. Namely, it is an ordered tree of elements so that leaves correspond to the fields in the interface, internal nodes correspond to groups or super-groups of the fields and the order among the siblings within the tree resembles the order of fields in the query interface. Moreover, the schemas for query interfaces can also be represented by using parenthesis notation. For example the schema in Fig. 6 and 7 can also be represented by (A (Author (First Name, Middle Name, Last Name), Title, Subject, ISBN, Publisher)) and (B (Author Name (First Name, Last Name), Name of Book, Topic, ISBN)) respectively. The unified query interface obtained from large number of query interfaces must satisfy following two types of constraints:

•Structural Constraints: A unified search interface is said to satisfy the structural constraint only if it preserves the ancestor-descendent relationship between the attributes in each of source interfaces.

•Grouping Constraints: It has been observed that certain groups of attributes appear together with high frequency in a set of interfaces over a domain of discourse. Thus, in the integrated interface these groups should appear together in order to satisfy the grouping constraints.

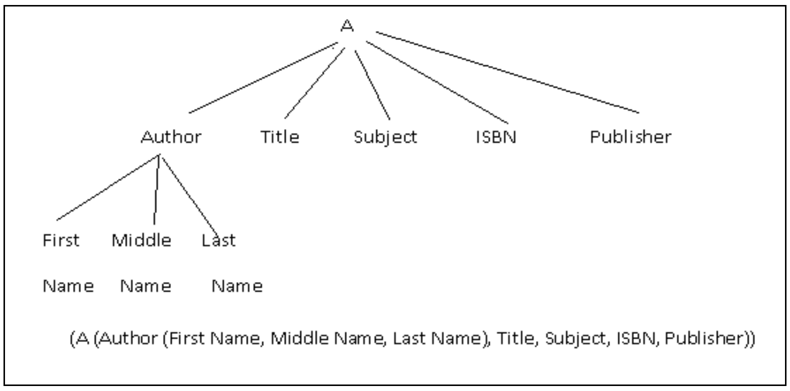

A
Author
Title
Subject
ISBN
Publisher
First Name
Middle Name
Last Name
(A (Author (First Name, Middle Name, Last Name), Title, Subject, ISBN, Publisher))

Fig. 6: Hierarchical representation of query interface A

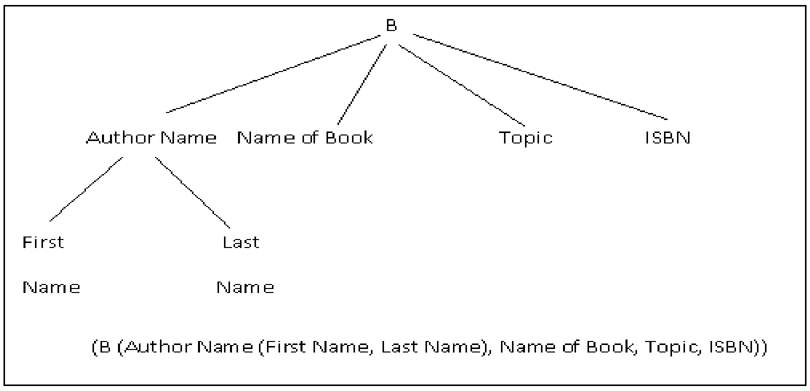

B
Author Name
Name of Book
Topic
ISBN
First Name
Last Name
(B (Author Name (First Name, Last Name), Name of Book, Topic, ISBN))

Fig. 7: Hierarchical representation of query interface B

Integrating Algorithm
The inputs of the integrating algorithm consist of:
1.A set of query interfaces in a particular domain,
2.Mappings that show the semantic mappings between the fields of different query interfaces.

The framework to obtain the semantic mappings from a given set of search interfaces has been proposed in. The implemented integration algorithm, in this paper, uses the semantic correspondences. As discussed, the Similarity Value Matrices (SVMs) contain the Estimated Similarity Values (ESVs) obtained after matching process. The mappings having estimated similarity value below threshold value would be ignored. For example, if an attribute author of first interface is matched with attributes first name, ISBN and subject of the second interface, the matcher returns estimated similarity values for each of three mappings i.e. for author and First Name, author and ISBN and for author and subject. The estimated similarity values of these three mappings are compared with a threshold value. The mappings having estimated similarity values greater than threshold value are treated as important mappings and are stored in Mapping Knowledge Base for future reference. Thus semantic mappings in the Mapping Knowledge Base between all the fields over the schema can be organized in form of clusters. Within a given domain, a cluster consists of all the fields of different schema that are semantically equivalent. For example, in Figure 6 and 7, author, Author Name are semantically equal and they are in same cluster, called Author. Similarly, Title and Name of Book are semantically same and they are also in the same cluster, called Title. Each cluster has been represented by a pair (Cluster Id, Cluster Name), where Cluster Id represents a unique Id for each cluster and Cluster Name represents the name of cluster having a particular Cluster Id. A pair of form (Schema Name, Field Name), where Schema Name represents the name of schema containing the field and Field Name uniquely identifies a particular field in a schema tree denotes the elements of a cluster. There may be schemas that do not have fields in all clusters. In that case, the cluster can have an entry (Schema Name, Null), where Null shows that the schema does not have any field in the cluster. In the integrating process, the potential mappings between the internal elements (nodes) are identified based on the mappings of their sub-elements which can be either leaf elements (nodes) or internal elements. The integrating of a set of schemas is carried out pairwise i.e. by integrating the first two schemas, S_1 and S_2. The resulted integrated schema, denoted by S_{12} is then further integrated with the third schema S_3, and so on.

Several functions are defined on the elements of trees as well as on the set of integrate operations to be applied by the algorithm for the integrating of schema. The implemented algorithm uses the following functions defined on elements of trees:

- relation (e, f): For attributes e and f, relation function returns 1 or 0. if there exists semantic relationship between e and f, the function returns 1, otherwise 0;
- combinesubtrees (e, f, ef): This function combines two subtrees e and f to give subtree ef ;
- assignname(ef) : This function assigns the most general name to the combined subtree. For example, if e = author and f = author name, then the function assignname assigns author name to the root of subtree ef as author name is more general than author;
- removedupnodes(ef): This function removes the duplicate nodes or the nodes from the subtree ef;
- insert(ef, Sij): insert(ef, Sij) function inserts the subtree ef in the integrated schema tree Sij.

An algorithm for the process of integrating two schemas, S_i and S_j uses the semantic mappings. If both schema trees are empty then empty schema is returned as the output, otherwise if one tree is empty then another tree is returned as the output schema. The main body of the algorithm is executed when both schema trees are not empty. This part is executed by finding the depths of both the trees. Both schema tree are integrated level by level. Intially, for each level of S_i , another tree Sj is scanned to check the semantic mappings. If there exists semantic mapping between a node e in tree Si and a node in tree S_j, then subtrees of nodes e and f are integrated. After integrating the subtrees, the integrated subtree is renamed and then duplicate elements are removed from the integrated subtree. The integrated subtree is then inserted in the integrated schema tree S_{ij}.

```
Algorithm integrate($S_i$ , $S_j$)
begin
  if $S_i$ and $S_j$ are empty trees then
    return empty tree;
  if $S_i$ is empty tree then
    return $S_{ij} = S_j$;
  if $S_j$ is empty tree then
    return $S_{ij} = S_i$;
  leveli = depth($S_i$);
  levelj = depth($S_j$);
  for i = 0 to leveli do
  begin
    for j = 0 to levelj do
    begin
      integratebylevel(i, j);
      return $S_{ij}$;
    end;
  end;
end;                                                contd.
contd.
Algorithm integratebylevel(i, j)
begin
  for each node e at level i in $S_i$ do
  begin
    for each node f at level j in $S_j$ do
    begin
      if relation(e, f) then
      begin
        combinesubtrees(e, f, ef);
        assignname(ef);
        removedupnodes(ef);
        insert(ef, $S_{ij}$);
      end;
    end;
  end;
end;
```

```
Algorithm integratebylevel(i, j)
begin
  for each node e at level i in $S_i$ do
  begin
    for each node f at level j in $S_j$ do
    begin
      if relation(e, f) then
      begin
        combinesubtrees(e, f, ef);
        assignname(ef);
        removedupnodes(ef);
        insert(ef, $S_{ij}$);
      end;
    end;
  end;
end;
```

The inputs of the integration algorithm consist of set of query interfaces and mapping definition between them that are obtained with high accuracy. A tree structure is used to represent each schema. For a particular set of search interfaces, the algorithm displays an integrated query interface in Fig.8. A problem specific to integrating query interfaces is that the unified interface may have too many fields to be user-friendly. To remedy these problem fields that are less important (e.g. they can be detected by frequencies of occurrences in the various schemas) can be trimmed from the integrated interface

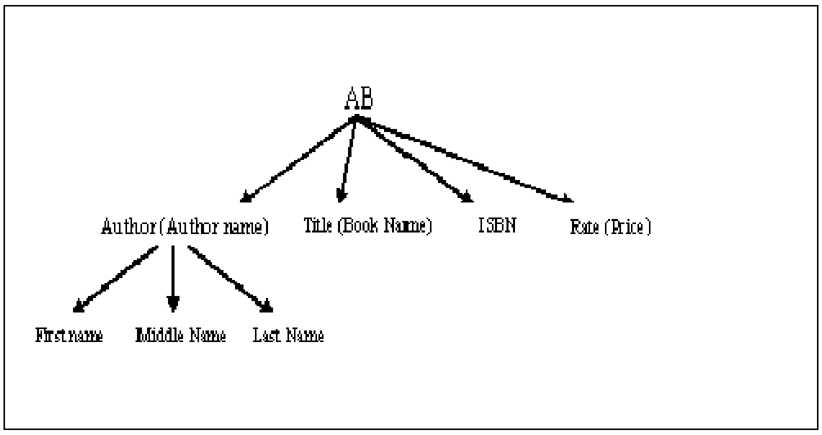


Fig. 8: The integrated query interface for the book domain

V. Implementation & Analysis

In implementation a set of query interfaces in a particular domain and mappings that show the semantic mappings between the fields of different query interfaces has been used as an input .Some interfaces are shown in the Figs.(9, 10, 11, 12) those have been taken as an input for the book domain.

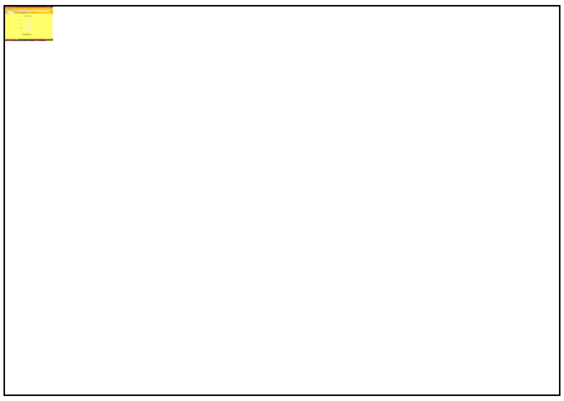


Fig. 9: Interface 1

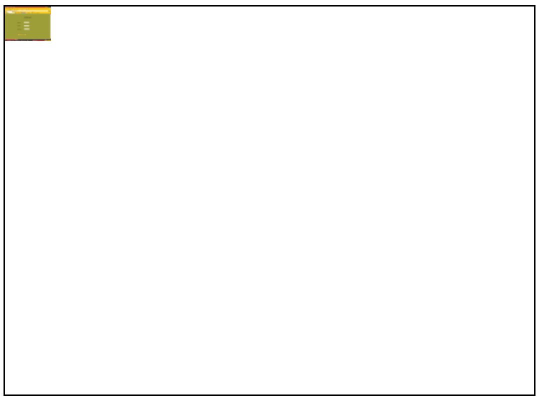


Fig. 10: Interface 2

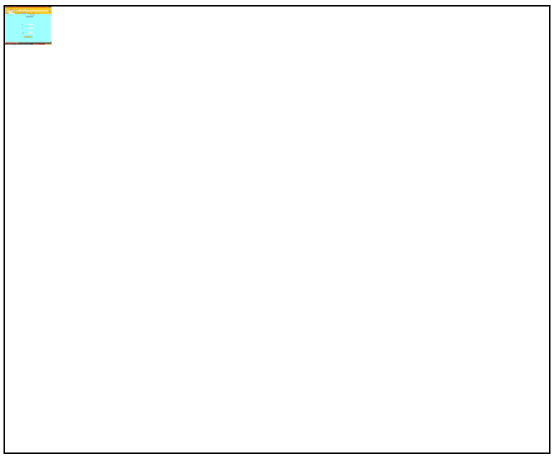
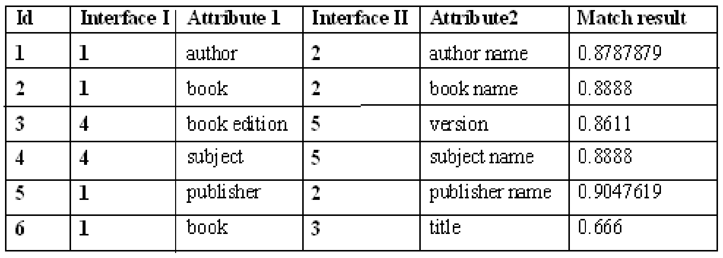


Fig. 11: Interface 3

Fig. 12: Interface 4

Fig. 13 shows a table which has the matching results of the attributes of the interfaces.These results are taken above a particular threshold which is 0.65 and the integrated interface by using these results is shown in Fig. 14.



Id	Interface I	Attribute 1	Interface II	Attribute2	Match result
1	1	author	2	author name	0.8787879
2	1	book	2	book name	0.8888
3	4	book edition	5	version	0.8611
4	4	subject	5	subject name	0.8888
5	1	publisher	2	publisher name	0.9047619
6	1	book	3	title	0.666

Fig. 13: Mapping results of attributes above threshold 0.65

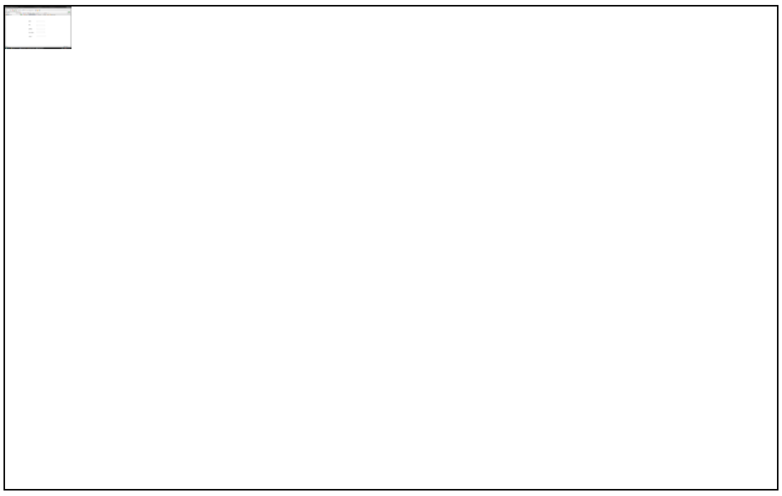


Fig. 14: Integrated interface

The performance of field matching and integrating interfaces has been measured via three metrics: precision, recall, and F-measure. Precision is the percentage of correct mappings over all mappings identified by the system, while Recall is the percentage of correct mappings identified by the system over all mappings as given by domain experts. Suppose the number of correctly identified mappings is C, the number of wrongly identified mappings is W and the number of unidentified correct mappings is M, then the precision of the approach is given by the expression given below

$$P = C/ (C + W) \quad (1)$$

and the recall, R, of the approach is

$$R = C/ (C + M) \quad (2)$$

F-measure incorporates both precision and recall. F-measure is given by

$$F = 2PR/ (P + R) \quad (3)$$

where: precision P and recall R are equally weighted.

For every domain average precision, average recall and average F-measure were computed by the expressions given below:

Average Precision = ? Pi /N (4)

Average recall = ? Ri / N (5)

Average F-measure = ? (F-measure)i /N (6)

Where N is total no. of interfaces matched and i range from 1 to N.

To implement the proposed work we have applied the proposed work on following four domains

- Book
- Airline
- Automobile
- Electronics

On calculating precision, recall and f-measure for the four domains (Book, Airline, Auto, Electronics) the result is been found can be seen in the following figures-

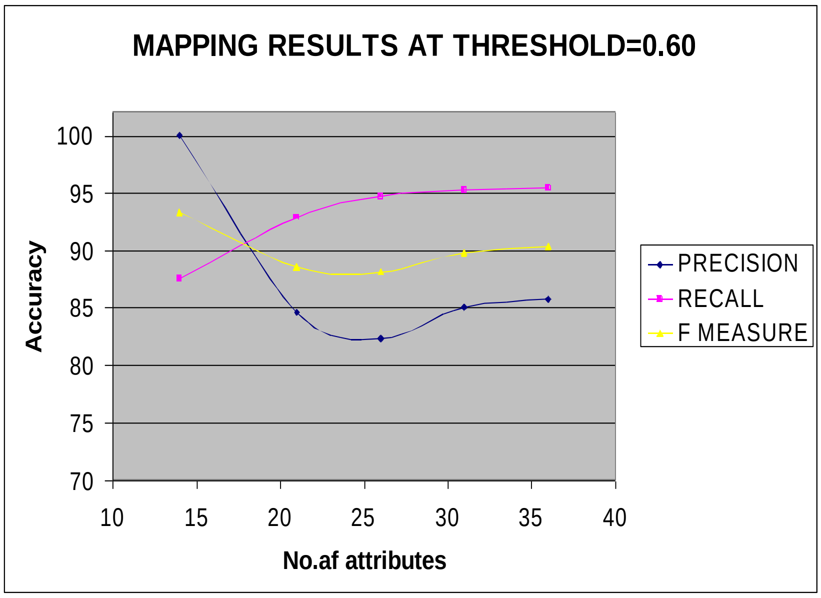


Fig. 15: Mapping results of book domain at threshold=0.60

The above curve shows the mapping results i.e. Precision, Recall and F-measure curves at threshold value 0.6 for book domain.

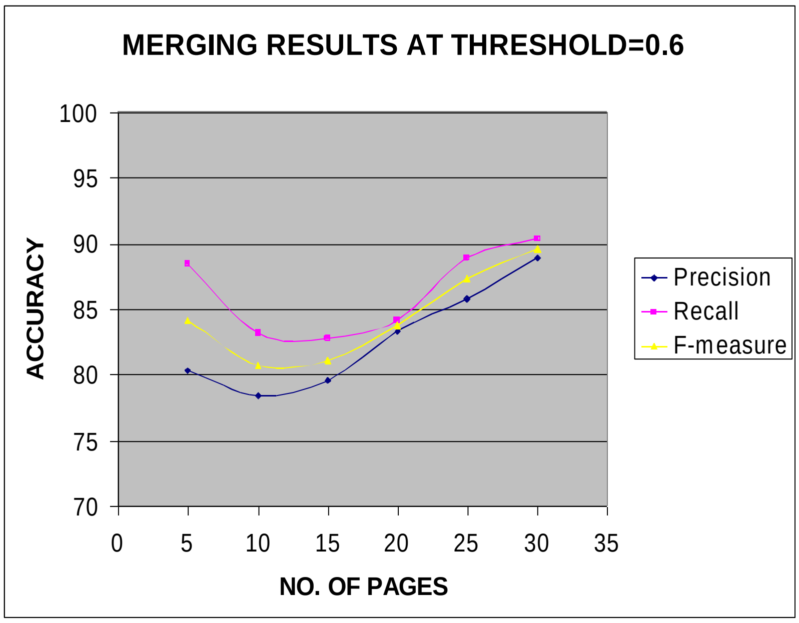


Fig. 16: Merging results of book domain at threshold=0.60

The above curve shows the merging results i.e. Precision, Recall and F-measure curves at threshold value 0.6 for book domain.

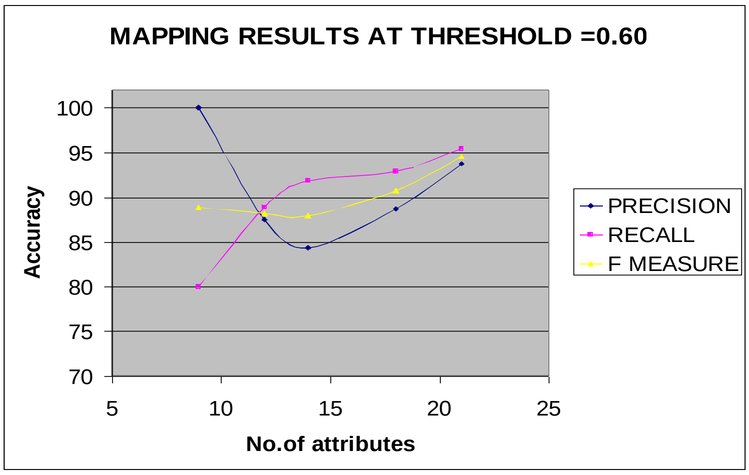


Fig. 17: Mapping results of airline domain at threshold=0.60

The above curve shows the mapping results i.e. Precision, Recall and F-measure curves at threshold value 0.6 for airline domain.

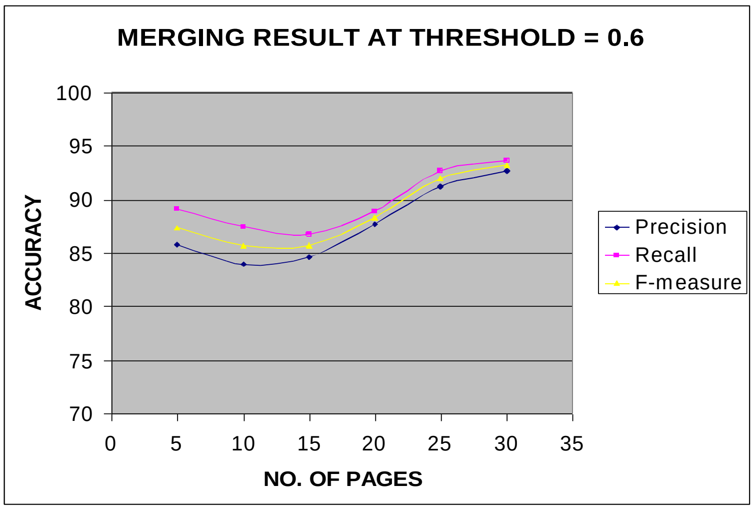


Fig. 18: Merging results of airline domain at threshold=0.60

The above curve shows the merging results i.e. Precision, Recall and F-measure curves at threshold value 0.6 for airline domain.

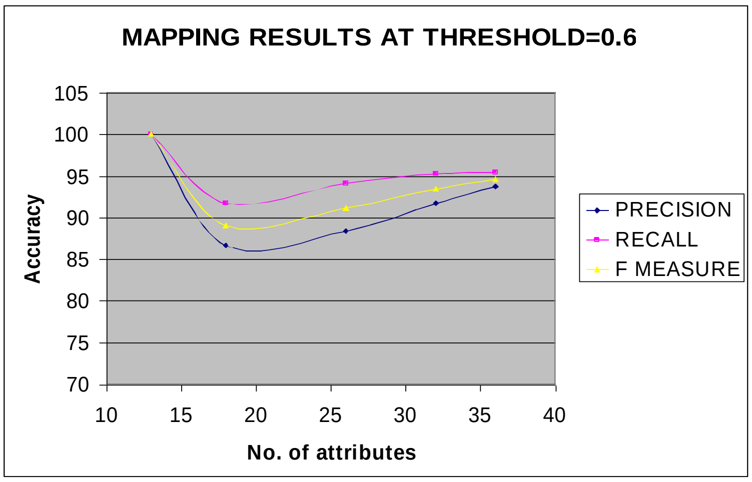


Fig. 19: Mapping results of auto domain at threshold=0.60

The above curve shows the mapping results i.e. Precision, Recall and F-measure curves at threshold value 0.6 for automobile domain.

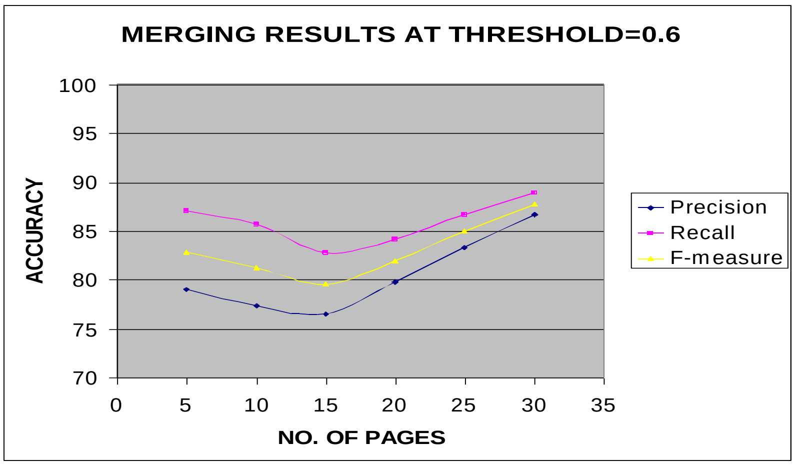


Fig. 20: Merging results of auto domain at threshold=0.60

The above curve shows the merging results i.e. Precision, Recall and F-measure curves at threshold value 0.6 for automobile domain.

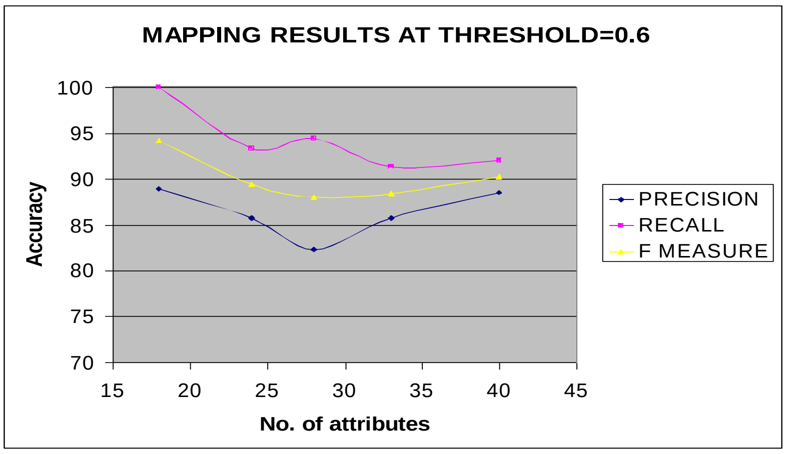


Fig. 21: Mapping results of electronics domain at threshold=0.60

The above curve shows the mapping results i.e. Precision, Recall and F-measure curves at threshold value 0.6 for electronics domain

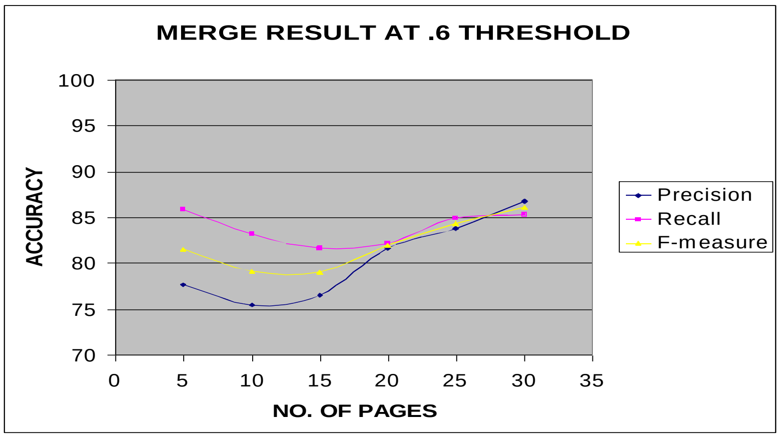


Fig. 22: Merging results of electronics domain at threshold=0.60

The above curve shows the merging results i.e. Precision, Recall and F-measure curves at threshold value 0.6 for electronics domain. The Threshold value0.6 was used here to compute Average Precision, Average Recall and Average F-measure for every domain and the results are tabulated in the Fig. 23, Fig. 25 and graphed in Fig. 24, Fig. 26.

Average mapping results-

DOMAINS	PRECISION	RECALL	F-MEASURE
BOOK	87.534	93.152	90.018
AIRLINE	90.858	89.816	90.07
AUTO	92.08	95.29	93.844
ELECTRONICS	86.214	94.014	90.008

Fig. 23: Average mapping results at threshold=0.60

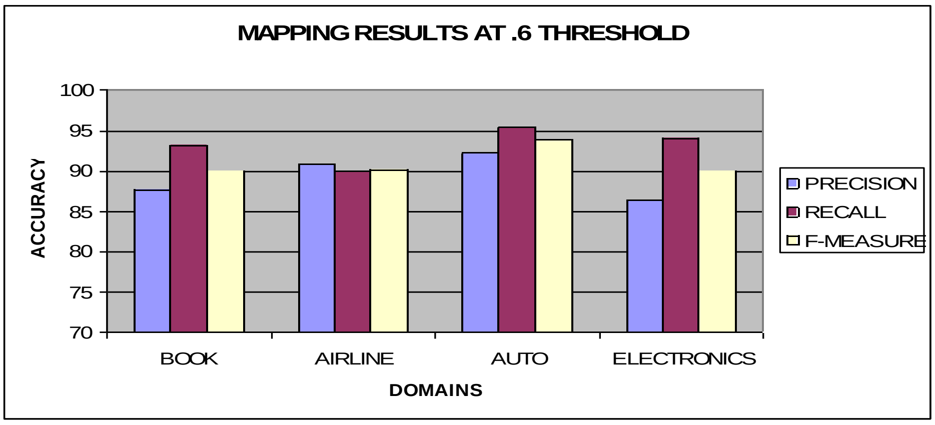


Fig. 24: Average mapping results at threshold=0.60

Average merging results –

DOMAINS	PRECISION	RECALL	F-MEASURE
BOOK	80.316	88.42	84.038
AIRLINE	85.756	89.128	87.394
AUTO	79.032	87.096	82.72
ELECTRONICS	77.628	85.852	81.36

Fig. 25: Average merging results at threshold =0.60

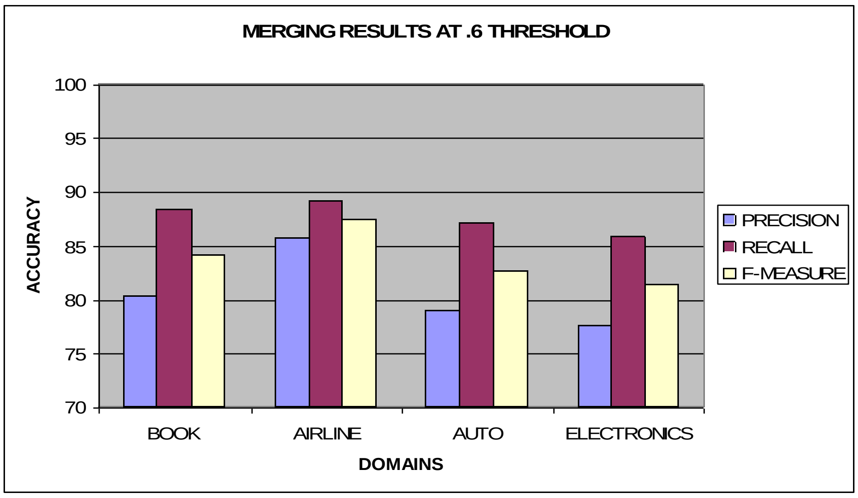


Fig. 26: Average merging results at threshold=0.60

VI. Conclusion

As a large amount of web contents residing on deep web are generated dynamically from databases and other data sources deep from the user. These web contents are generated only when queries are asked via a search interface, rendering interface integration a critical problem in many application domains, such as: semantic web, data warehouses, e-commerce etc. Many different integration solutions have been proposed so far. In this paper, a novel approach for interface integration has been implemented. It uses semantic mappings to integrate the search interfaces in a particular domain of interest. The contribution in this paper is an implementation of an automatic approach to the problem of integrating large-scale collections of query interfaces of the same domain. The implemented method transforms a set of interfaces in the same domain of interest into a global interface such that all ancestor-descendant relationships (structural constraints) are preserved and the grouping constraints are satisfied as much as possible.

To the best of our knowledge, this is the first piece of work which makes such a guarantee for interface integrating. Experiments are performed in four domains to demonstrate that the integrated interface produced by the implementation of proposed algorithm in each of these domains is natural. The results obtained for various domains are highly accurate as the accuracy obtained for a threshold value of about 0.60 is about 85% and the accuracy reaches the value about 90% for the threshold value of 0.65.To the best of our knowledge, this is the first piece of work which makes such a guarantee for interface integrating. Implementation has been performed on four domains and the results are found highly accurate.

References

[1]. A. K. Sharma, Komal Kumar Bhatia: "Automated Discovery of Task Oriented Search Interfaces through Augmented Hypertext Documents" accepted at First International Conference on Web Engineering & Application (ICWA2006).
[2]. A.K.Sharma and Komal Kumar Bhatia, A Framework for Domain-Specific Interface Mapper (DSIM)
[3]. AnHai Doan:Learning to Map between Structured Representations of Data. PhD thesis, University of Washington..
[4]. Luciano Barbosa, Juliana Freire, "Combining Classifiers to Identify Online Databases", WWW2007/track
[5]. T. Mitchell. Machine Learning. McGraw Hill, 1997
[6]. L. Barbosa and J. Freire. "Searching for Hidden-Web Databases". In Proceedings of WebDB, pages 1–6, 2005
[7]. http://en.wikipedia.org/w/index.php?title=Deep_web&redirect=no
[8]. A. K. Sharma, J. P. Gupta, "An Architecture of Electronic Commerce on the Internet", accepted for the publication in the fourth coming issue of Journal of Continuing Engineering Education, Roorkee, Jan 2003
[9]. Dinesh Sharma, A.K. Sharma, Komal Kumar Bhatia, "Web crawlers: a review", NCTC-2007
[10]. Dinesh Sharma, A.K. Sharma, Komal Kumar Bhatia," Search engines: a comparative review",NGCIS-2007
[11]. http://www.cs.utah.edu/~juliana/pub/webdb 2005. pdf
[12]. http://en.wikipedia.org/wiki/Deep_web
[13]. www.invisibleweb.com
[14]. Alexandros Ntoulas Petros Zerfos Junghoo Cho, "Downloading Hidden Web Content", UCLA
[15]. A. Bergholz and B. Chidlovskii. Crawling for Domain-Specific Hidden Web Resources. In Proceedings of WISE, pages 125–133, 2003
[16]. http://en.wikipedia.org/wiki/parsing
[17]. http://www.brightplanet.com/imagesstories.pdf/deepwebwhitepaper.pdf
[18]. http://amazon.com
[19]. www.cs.utah.edu/~juliana/pub/freire-sbbd2004.pdf
[20]. http://www2007.org/papers/paper429.pdf
[21]. http://www.lib.berkeley.edu/TeachingLib/Guides/Internet/InvisibleWeb.html
[22]. http://www.weblens.org/invisible.html
[23]. http://websearch.about.com/od/invisibleweb/a/invisible_web.htm
[24]. http://www.searchengineshowdown.com/features/av/review.html
[25]. http://harvest.sourceforge.net/harvest/doc/index.html
[26]. www.searchengineshowdown.com/features/google/review.html
[27]. http://infolab.stanford.edu/~backrub/google.html#hits
[28]. http://www.asis.org/annual-96/ElectronicProceedings/chu.html
[29]. http://daphne.palomar.edu/TGSEARCH/
[30]. The Overview of Web Search Engines written by Sunny Lam
[31]. Google: Main Page. http://www.google.com, 2000.
[32]. L. Gravano, K. Chang, H. Garcia-Molina, and A. Paepcke. STARTS: Stanford
[33]. Proposal for Internet Meta-searching. Proc. ACM SIGMOD Conference, pages 207-218, 1997. S. Brin and L. Page. The Anatomy of a Large-Scale Hypertextual Web Search Engine. Proc. 7th WWW Conference, 1998.
[34]. J. Rennie and A. McCallum, "Using reinforcement learning to spider the web efficiently," in Proc. International Conference on Machine Learning (ICML), 1999.
[35]. http://www.almaden.ibm.com/almaden/feat/www8/
[36]. http://softbase.uwaterloo.ca/~tozsu/courses/cs856/W05/Presentations/Mohamed.pdf
[37]. http://www10.org/cdrom/posters/p1049/index.htm
[38]. http://dbpubs.stanford.edu:8090/cgi-bin/makehtml.cgi?document=2001/19& format= &page=4
[39]. http://cis.poly.edu/tr/tr-cis-2001-03.pdf
[40]. S. Lawrence and C. L. Giles. Context and Page Analysis for Improved Web Search.IEEE Internet Computing, 2(4):38-46, 1998.
[41]. Fausto Giunchiglia and Pavel Shvaiko: Semantic Matching. In the Knowledge Review journal, 18(3):265-280,2004.
[42]. Hong-Hai Do, Erhan Rahm, COMA-A system for flexible combination of schema matching approaches. In Proc. 28th VLDB

Conference.
[43]. Erhard Rahm, Philip A. Bernstein. A survey of approaches to automatic schema matching. VLDB Journal 10, 2001.
[44]. Z. Zhang, B. He, and K. Chang. Understanding Web Query Interfaces: Best- Effort Parsing with Hidden Syntax. SIGMOD Conference, 2004.
[45]. Jayant Madhavan, Philip A. Bernstein, Erhard Rahm, Generic Schema Matching with Cupid, VLDB 2001.
[46]. [46] AnHai Doan, P. Domingos, A. Halevy. Reconciling Schemas of Disparate Data Sources: A Machine-Learning Approach. In SIGMOD Record, 2001.
[47]. Sergey Melink, Hector Garcia-Molina, Erhard Rahm. Similarity Flooding: A Versatile Graph Matching Algorithm and its Application to Schema Matching. In Proc. 18th International.Conf. On Data Engineering, San Jose CA, 2002.
[48]. Li W, Clifton C, Liu S (2000) Database integration using neural network: implementation and experiences. Knowl Inf Syst 2(1): 73-96.
[49]. M. Dell'Erba, O. Fodor, F. Ricci, H. Werthner:. Harmonise: A Solution for Data Interoperability. IFIP I3E 2002.
[50]. Do H.-H., Melnik S., and Rahm E.: Comparison of Schema Matching Evaluations, Proc. GIWorkshop "Web and Databases", Erfurt, Oct. 2002.
[51]. Mike Burner, "Crawling towards Eternity: Building an archive of the World Wide Web" Web Techniques Magazine, 2[5], May 1997.
[52]. Alexandros Ntoulas Petros Zerfos Junghoo Cho, "Downloading Hidden Web Content", UCLA Computer Science, fntoulas, pzerfos, chog@cs.ucla.edu
[53]. Searching for Hidden Web Databases Luciano Barbosa, University of Utah, lab@sci.utah.edu, Juliana Freire, University of Utah, juliana@cs.utah.edu
[54]. Focused crawling: a new approach to topic specific Web resource discovery,Soumen Chakrabarti, Martin van den Berg , Byron Domc
[55]. Brian Pinkerton, "Finding what people want: Experiences with the web crawler", Proc of WWW Conf. , 1994.
[56]. England Jared Cope Nick Craswell and David Hawking, "Automated Discovery of Search Interfaces on the Web", Fourteenth Australasian Database Conference (ADC2003).
[57]. Ipeirotis, P., Gravano, L. & Mehran, S. (2001), 'Probe, count, and classify: categorizing hidden web databases', ACM SIGMOD 30(2), 67 –78.
[58]. Dreilinger, D. & Howe, A. E. (1997), "Experiences with selecting search engines Using metasearch", ACM Transactions on Information Systems 15(3), 195–222.
[59]. B. He, Z. Zhang, K. Chang, "Knocking the Door to the Deep Web: Integration of Web Query Interfaces", SIGMOD Conference, Demo, 2004.
[60]. www.w3.org/hypertext/WWW/MarkUp/MarkUp.html -- official HTML specification.
[61]. W. Wu, C. Yu, A. Doan, and W. Meng, "An Interactive Clustering-based Approach To Integrating Source Query Interfaces on the Deep Web", SIGMOD Conference, 2004.
[62]. www.searchengineshowdown.com/features/yahoo!search/review.html
[63]. http://en.wikipedia.org
[64]. http://www.devbistro.com/articles/Misc/Implementing-Effective-Web-Crawler
[65]. http://edtech.boisestate.edu/bschroeder/publicizing/three_types.htm
[66]. http://www.yutrend.com/topics/basics/
[67]. http://www.sc.edu/beaufort/library/pages/bones/lesson1.shtml
[68]. http://en.wikipedia.org/wiki/Web_search_engine
[69]. www.searchengineshowdown.com/features/ask/review.html
[70]. www.searchengineshowdown.com/features/gigablast/review.html