ISSN XXXX XXXX © 2017 IJESC

Research Article Volume 7 Issue No.6

Design and Implementation of Automatic Extraction of Elements of Web Search Interfaces for LVS Table in HIWE

Anil Kumar[1], Dr. Mukesh Rawat[2]
M.Tech Student[1], Associate Professor[2]
Department of CS/IT
MIET, Meerut, India

Abstract:
The World Wide Web is a huge source of hyperlinked information. Web is growing every moment in context of web documents. Search engines use web crawlers to collect web documents from web for storage and indexing. In the existing system, it has been observed that the repository resources are limited. So it has become an enormous challenge to manage the local repository (storage module of search engine) in a way to handle the web documents efficiently that leads to less access time of web documents and proper utilization of available resources. 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. These pages are often referred to as hidden web. Search engines cannot discover and index such pages as they have no static links. However, according to recent studies, the content provided by many Hidden Web sites is often of very high quality and can be extremely valuable to many users. A large amount of web contents residing on hidden web are generated dynamically from databases and other data sources hidden 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 thesis, a novel approach for extracting search interfaces in a particular domain of interest is given and the tools used in implementation are JDK 1.5 (Java.io and Java.net packages).

1. INTRODUCTION

The World Wide Web (WWW) is an Internet client communication system for retrieving and displaying multimedia hypertext documents. Due to extremely large number of pages present on Web, the search engine depends upon crawlers for the collection of required pages. A crawler follows hyperlinks present in the documents to download and store the pages in a database. Generally the documents are static documents in the sense that these are written and translated into hypertext markup language (HTML) and their contents remain constant over time. Such documents provide information about services, charter and capabilities of an organization. But, for the most part, hypertext document consist of information that gets updated on a weekly, monthly or even yearly basis. This type of information is referred as dynamic web information. Recent studies indicate that tremendous amount of content on the Web is dynamic, though current day crawlers only crawl the publically index able Web [1]. In fact much of the web content (both static and dynamic) remain inaccessible i.e. the pages are essentially "hidden" from a typical web user. These pages are often referred to as the Hidden Web [3]. According to many studies, the size of the Hidden Web is increasing rapidly as more organizations put their valuable content online through an easy-to-use Web interface [3]. It is estimated that the hidden web contains anywhere between 7500 and 91850 tetra bytes of information. Moreover, the content provided by many Hidden-Web sites is often of very high quality and can be extremely valuable to many users. The hidden web content continues to grow with the time. Therefore there has been increased interest in retrieval and integration of hidden-web data with a view to leverage high quality information available in online database. Although previous works have addressed many aspects of integration including matching form schemata and automatically filling out the forms, the problem of locating relevant data sources has been largely overlooked. It is crucial to dynamically discover these resources due to the constant change in the data sources.

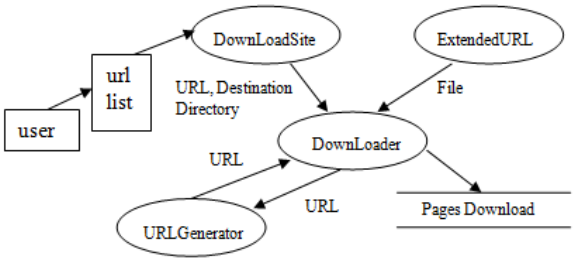
Crawling the hidden Web is a very challenging problem for three fundamental reasons:
1. Scaling ability
2. The need for crawlers to handle search interfaces designed primarily for humans.
3. Mechanism for crawler to find domain specific search interface.

Since the search interfaces acts as the entry points for the hidden web contents, it becomes very important to identify these search forms efficiently. Although there exists large number of techniques to correctly identify the search interfaces, the current existing techniques parse each web document regardless of whether the web page is search interface or not. In this paper, we proposed a mechanism for extracting forms from downloaded pages which is further used to extract domain specific search interface by adopting domain-specific-assisted approach for crawling the hidden web. Also, a framework that allows hidden web crawler to automatically find domain specific search interfaces has been proposed. In this paper, architecture has been proposed that identifies the form which is further used to identify search interfaces. Since search forms are the entry-points into the hidden Web, a hidden web framework to automatically identify forms on web and store them into repository. The repository would be further used for identify search interfaces and crawling the hidden web contents. The given architecture first takes a seed URL and downloaded all the pages for a given

website by finding the hyperlinks in a page and download the page referred by the hyperlink and store them in the repository, then this repository is used to extract pages contains forms and identify search forms among them. A mechanism for identifying the search forms available on the web portal and fetching their corresponding elements has been proposed. The proposed mechanism has the following functional components:

1.1 Component of proposed system

DFD1:



The following modules are used to extract the query interfaces from the WWW –

Download Site

In this module the seed URL (whose pages have to be downloaded) and the destination directory (where all the downloaded pages will be stored) are given.

Downloader

This module download the page and separate out the Hyperlinks from the page and stored them in the Vector. The algorithm used by the Downloader is shown in Fig 1.1

```
        Downloader()
        {
startDownload()
        }
        startDownload()
        {
        fdir← getDirectory()
        if(fdir not exists)
mkdirs()
        ffile← getFile()
        if(ffile not exists)
ffile← index.html
        else
ffile←filename
        listOfURL←downloadAndFillVector(InputStream,Outp
utStream)

        while(listOfURL not empty)
startDownload(each URL in listOfURL)
        }

        downloadAndFillVector(InputStream,
OutputStream)
        {
        while(characters read from ffile !=EOF)
if(hyperlink)
v← hyperlink
else
write(characters)
        return formvectorOfURLs(v)
        }
        formvectorOfURLs(v)
        {
                return getURLs()
        }
```

ExtendedURL

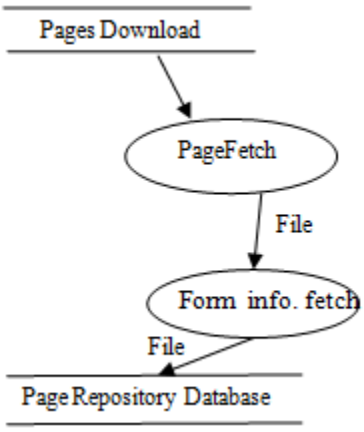
This module checks whether the file mentioned in the URL is a directory or a file, if no file is mentioned in the URL it will set the file as "index.html". The algorithm used by the ExtendedURL is shown in Fig 4.2.2

```
        getFile()
        {
                return file
        }
        getDirectory()
        {
                return dir
        }
```

URL Generator

This module fetches a URL from vector and provided it to the Down Loader module. The algorithm for the URL Generator is implemented by the function **getURLs ()**. The algorithm used by the URL Generator is shown in Fig 1.2

DFD2:



PageFetch

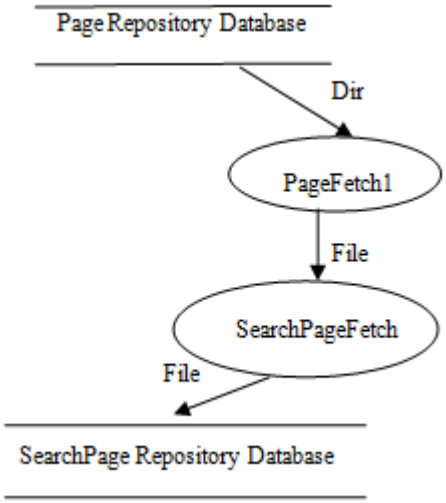
This module fetches the files from the Pages Download in the data store. The algorithm for the PageFetch is implemented by the function **extract (URL of Data Store)**. The algorithm used by the **PageFetch** is shown in Fig 4.2.4

- **For Info Fetch**

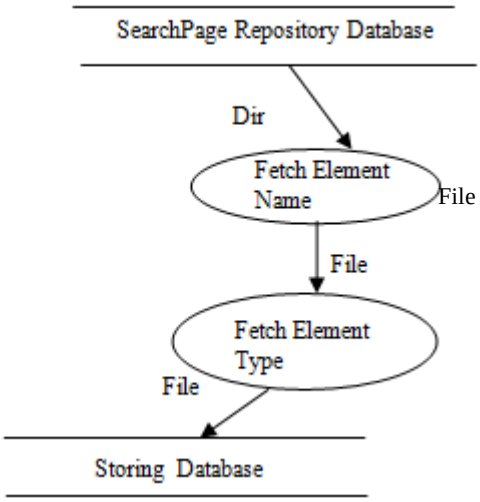
This module First of all check whether the fetched file contain any form tag or not. If the fetched file contains a form tag, then copies the information within the form tag into a file with the name same as the name of the fetched file and stored it into a Repository Database. The algorithm for the **Form Info Fetch** is implemented by the function **ex_form (URL of the html file in Data Store)**. The algorithm used by the **Form Info Fetch** is shown below-

```
        ex_form (sr)
        {
        s_chk←characters in file represented by sr
        if ("<form" found in s_chk)
                write characters between "<form>" and
"</form>" in a file
        }
```

DFD3:



DFD4:



This module check whether the fetched file contain the tag indicating any element such as textbox aur combo box etc. If the fetched file contains a form tag, then copies the information into a file with the name same as the name of the fetched file and stored it into a Storing Database. The algorithm for the **Fetch Element Name (String Tagtype)**

2. ANALYSIS

The performance of extracting forms and identifying search interfaces among them has been measured via three metrics: precision, recall, and F-measure. *Precision* is the percentage of correctly identified search interfaces over all the search interfaces identified by the system, while *Recall* is the percentage of correctly identified search interfaces by the system over all the correctly identified search interfaces and unidentified correct search interfaces. Suppose the number of correctly identified search interfaces is C, the number of wrongly identified search interfaces is W and the number of unidentified correct search interfaces 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. In implementation the proposed work has been applied on book domain. On calculating precision, recall and f-measure for Book domain the result is been found can be seen in the following table-

Pages	C	W	M	P	R	F
5	2	0	2	1.0	0.5	0.666667
10	5	0	4	1.0	0.555556	0.714286
15	7	0	6	1.0	0.538462	0.7
20	9	1	6	0.9	0.6	0.72
25	9	1	8	0.9	0.529412	0.666667
30	9	1	9	0.9	0.5	0.642857
35	11	2	10	0.846154	0.52381	0.647059
40	13	3	13	0.8125	0.5	0.619048
45	16	2	13	0.888889	0.551724	0.680851
50	17	2	13	0.894737	0.566667	0.693878

The figure below shows the result for the Book domain based on precision, recall and
F-measure-

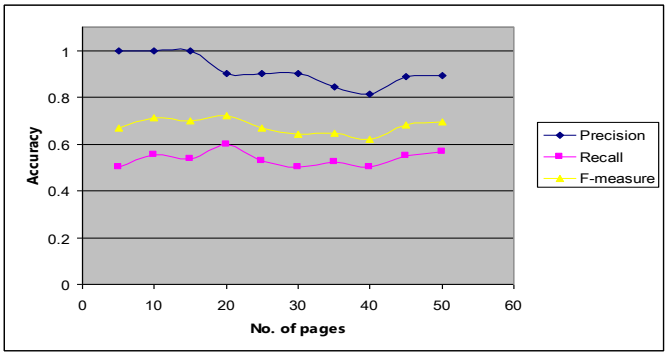


Figure1. Performance of Proposed System

The variation in the Precision, Recall and F-measure as Precision varies from 81% to 100%, Recall varies from 50% to 60% and F-measure varies from 62% to 72% is due to because some of the search interfaces are not correctly identified and some of the interfaces are not identified because they contain image icons in place of button and the system being proposed is unable to recognize these image icons. The figure below shows the form that is not correctly identified and The figure below shows the form that is not correctly identified and the form that is not identified as Fig 4.3.1 and Fig 4.3.2 respectively. The search form as shown in Fig 4.3.1 is not correctly identified because the labels of the text fields are not visible as the font color of the labels are white and the search form as shown in the Fig 4.3.2 is not identified because it contains image icon in place of button.

II. CONCLUSION

As a large amount of web contents residing on hidden web are generated dynamically from databases and other data sources hidden from the user. These web contents are generated only when queries are asked via a search interface, extracting the search interfaces is a critical problem in many application domains, such as: semantic web, data warehouses, e-commerce

etc. Many different solutions have been proposed so far. In this thesis, a promising approach for extraction of search interfaces has been implemented. It reads the caption of the button and if the caption of the button contains the text such as "go, search, find" etc. than such form is considered as a search interface. The results obtained for specific domain (Book Domain) is highly accurate as the Precision varies from 81% to 100%, Recall varies from 50% to 60% and F-measure varies from 62% to 72%. To the best of our knowledge, this is the first piece of work which makes such a guarantee for search interface extractor. Implementation has been performed on Book domain and the results are found highly accurate.

III. REFERENCES

[1]. A. K. Sharma, Komal Kumar Bhatia: "Merging query interfaces in domain specific hidden web databases" .

[2]. A.K.Sharma and Komal Kumar Bhatia, A Framework for Domain-Specific Interface Mapper (DSIM)

[3]. 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).

[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 Web DB, 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