Design of a Parallel and Scalable Crawler for the Hidden Web

Sonali Gupta, J. C. Bose University of Science and Technology, India
Komal Kumar Bhatia, J. C. Bose University of Science and Technology, India

ABSTRACT

The WWW contains a huge amount of information from different areas. This information may be present virtually in the form of web pages, media, articles (research journals/magazine), blogs, etc. A major portion of the information is present in web databases that can be retrieved by raising queries at the interface offered by the specific database and is thus called the hidden web. An important issue is to efficiently retrieve and provide access to this enormous amount of information through crawling. In this paper, the authors present the architecture of a parallel crawler for the hidden web that avoids download overlaps by following a domain-specific approach. The experimental results further show that the proposed parallel hidden web crawler (PSHWC) effectively and efficiently extracts and downloads the contents in the hidden web databases.

KEYWORDS

Domain-Specific, Focused Crawler, Form Processing, Hidden Web Crawler, Parallel Crawler, Scalable Crawler, Search Engine, Topic-Specific, URL Distribution, Web Page Classification

INTRODUCTION

Due to the colossal size of the WWW, search engines have become the imperative tool to search and retrieve information from it (Lawrence & Giles, 1998). The WWW can be divided into two parts: The Surface Web (or Publicly Index able Web) and the Hidden Web. The Surface Web includes the information on the surface of the Web reachable to the crawler purely by following hyperlinks whereas the contents of the Hidden Web are generally stored in the Web database and is available through search form interfaces. The Hidden Web is estimated to be significantly larger and contain much more valuable information than the "Surface Web or PIW" (Bergholz & Chidlovskii, 2003; Ntoulas et al., 2005).

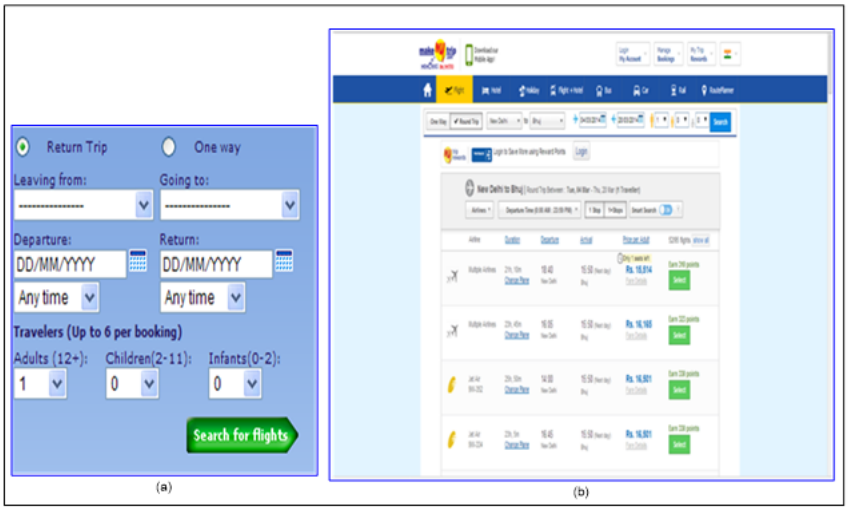
Accessing the contents of the Hidden Web is often time consuming and frustrating for an average user as he/she has to make queries through all potentially relevant Hidden Web databases (Madhavan et al., 2008; Sharma, 2008).

For example, if a user wants to search information about some flight, then in order to get the required information, he/she must go to airline site and fill the details in the search form acting as an interface to the web database. As a result he/she gets the details of the flights available. These types of pages are often referred to as dynamic or hidden web pages.

Figure 1(a) shows an example of such a search form that offers a search over the flights between two cities. Figure 1(b) shows an example of a dynamic or hidden web page.

DOI: 10.4018/IJIRR.289612

Figure 1. A typical search form and a dynamic (or hidden) web page



The crawlers employed by the conventional search engines are not intelligent enough to penetrate through the search interfaces and extract information from the underlying databases and hence does not allow the search engine to create an index from it. Move in the Web structure from a hyperlinked graph to form based search interfaces present the biggest challenge to a crawler's capability.

To help users better access the Hidden Web, efforts have been made on building intelligent crawlers that can pass through these search interfaces by automatically assigning suitable values to their form fields. But, visiting this huge fraction (80% of the World Wide Web) under the specified time bounds demand efficient and intelligent crawling techniques. A single crawling process cannot scale to the hundreds of thousands of Hidden Web databases on the WWW (Cho & Garcia-Molina, 2002; Diligenti et al., 2000). An alternate to sequential crawling is to have multiple crawling processes in parallel, where each such process loads a distinct page in tandem with others. Thus, the main concern of our paper lies in developing a parallel crawler that can retrieve the contents from both the Surface as well as the Hidden web.

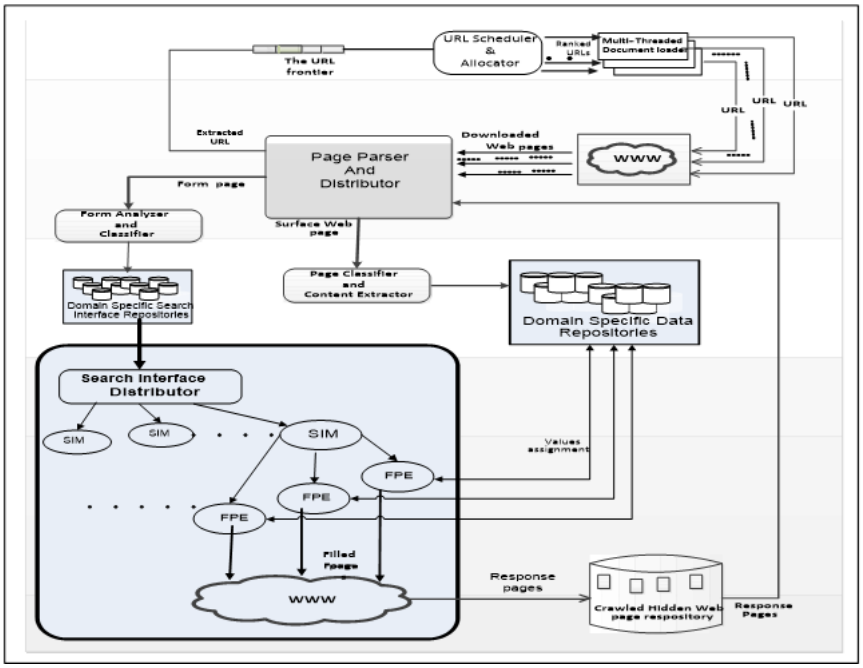
So, a parallel architecture of the Hidden Web crawler that seems to be an improved option in comparison to a single-process crawler has been proposed in this paper. It offers the following advantages:

- The proposed Parallel Hidden Web crawler not only effectively but also efficiently extracts and download the contents in the Hidden web databases.
- The proposed crawler minimizes the communication overhead needed to coordinate the parallel processes along with reduced network bandwidth consumption by adopting a domain-specific approach that overcomes the problem of heterogeneity across numerous domains.

SYSTEM ARCHITECTURE

Figure 2 presents the design of the proposed crawler in detail. In order to maximize the benefits, the design of the proposed Parallel and Scalable Hidden Web Crawler, PSHWC employs a number of components that work in parallel to offer scalability.

Figure 2. Proposed Architecture of PSHWC



The proposed Hidden Web crawler, PSHWC consists of the following functional components.

URL Scheduler and Allocator

A web crawler is typically initialized by some set of URLs called the seeds in the URL frontier. In a parallel crawler, the seed URLs must be overseen by all the parallel processes to achieve scalability and efficiency. This is done by partitioning and organizing the seed URLs into different Domain-Specific URL queues that can be accessed in parallel by the various parallel components of the crawler. For example, the URLs like www.makemytrip.com, https://placetoseeindelhi.wordpress.com etc. which provide information in Travel domain are included in the URL pool meant to store the URLs for Travel domain. The URL Scheduler & Allocator is responsible for organizing the seed URLs of the URL Frontier into the several Domain-Specific URL Queues for efficient crawling by the parallel processes.

To gather the seed URLs the proposed crawler takes advantage of the classification hierarchy offered by DMOZ directory. For each category in the hierarchy the system supports the retrieval of top N relevant URLs. Those top N URLs will serve as starting points for the crawler. Later as the crawler progresses, all the URLs that are gathered during crawling are added to these different Domain-Specific URL queues with the help of other components of the crawler.

The URL Scheduler and Allocator extracts the URLs from each of these URL Queues and assigns them to the multiple threads of the Document Loader for down-loading the associated web pages. This is done to enable parallel downloading of web pages in each domain. The URL Allocator dynamically chooses a URL from each Domain-Specific URL Queue for processing on the basis of the First-Come-First-Serve (FCFS) policy.

Multi-Threaded Document Loader

The Multi-threaded Document Loader is a high performance asynchronous HTTP client capable of downloading several web pages in parallel, initiates a number of downloader instances equal to the

Figure 3. The Page Parser and Distributor Algorithm

```
Page_Parser_Distributor()

Input: downloaded web pages.
Output: Type of the web page.

Start:
For each web page

{  if (page contains <FORM>)
            mark it as a form page for entry to the Hidden Web
             and pass to phase IV ;
    else
          classify as PIWP and pass to phase III;
}
```

number of URLs received (for downloading) from the URL Allocator by processing the different prioritized URL queues. The instances download the pages in parallel from the multiple web servers and pass them to the Page Parser and Distributor for further processing. If any information about the do-main of the just downloaded Web page has been made available in the due course, the Multi-threaded Document Loader also passes it to the Page Parser and Distributor along with the webpage.

Page Parser and Distributor

The web pages downloaded by the Multi-threaded Document Loader are passed to the Page Parser and Distributor, PPD. These web pages may contain search forms. Thus, each downloaded web page is given as input to one of the various threads of the Page Parser and Distributor which segregate the Publicly Indexed web pages (PIWP) and the Form Pages (FP). To distinguish the form pages, the Page Parser and Distributor searches the <FORM> tag in the HTML code of the downloaded web page. If the page does not contain the <FORM> tag, then it is assumed that the page can't act as an entry point for the Hidden Web and is a PIWP. However, if the page contains the <FORM> tag, then this may be treated as entry point for the Hidden Web (FP). This can be represented by the algorithm in Figure 3.

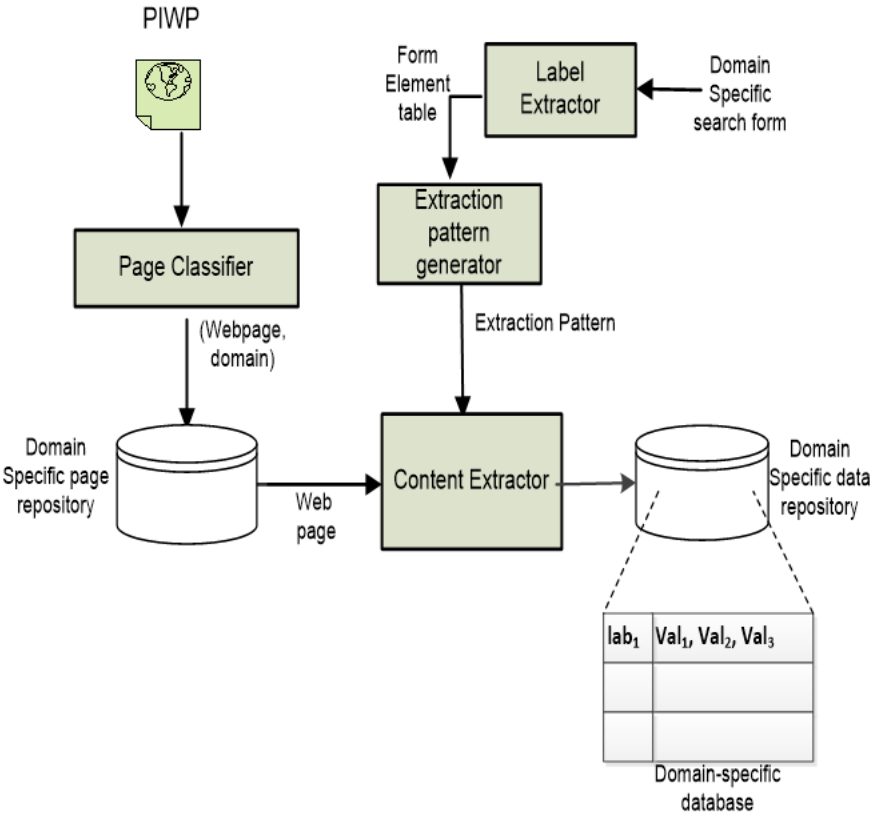
Page Classifier and Content Extractor

After separating the PIWPs and the FPs, the PIWPs are passed to the Page Classifier and Content Extractor (PCCE) that analyses each PIWP to create domain-specific repositories. It is responsible for storing the web pages and other useful data organized according to their domains. The repositories will be helpful for filling the search forms in order to download the Hidden Web contents so it is a very important component of the proposed parallel hidden Web Crawler.

Thus, PCCE performs the following three functions:

1. It takes the PIWPs and predicts domain of those web pages for which the domain information was not available.
2. Based on the domain information, it classifies them into different domains (Books, Entertainment, Food, Real Estate, Sports & Travel and stores them in the Domain-Specific Page Repositories (DSPRs).
3. After classification, domain-specific data repository (DSDR) for each specified domain is created by extracting useful content from the respective DSPR These DSDRs consist of Domain-Specific Databases that are further used by the Hidden Web Crawler to fill the search forms.

Figure 4. Framework of the Page classifier and Content Extractor for the creation of Domain-Specific Data Repository



The framework in Figure 4 shows the functionality of Page Classifier and Content Extractor to generate the Domain-Specific Data Repository (DSDR).

To predict the domain of the webpage, an instance of the system described in (Gupta & Bhatia, 2013a) has been used. The system is based on a Neural Network model classifier which works on a two-step framework: training the neural network (where it is given a set of web pages with known domains); testing (where the trained neural network is now used to predict the domain of a new webpage whose domain is unavailable). Training is done by extracting the keywords from the <META> and <TITLE> tags and assigning initial weights to them. These weights are adjusted so that the network works well on any unseen web pages. During this training of the neural network model the system also generates domain definitions for each specified domain by selecting the important keywords for that domain. The architecture of the Page Classifier is shown in Figure 5.

The generated DSPRs are then processed to create the DSDRs. Each DSDR maintains a database that defines the possible attributes and their values for filling the search forms in that domain. As an example, consider the Database in Table 1 for the Travel domain consisting of attributes and values.

A set of extraction patterns is generated by the Extraction Pattern Generator (EP Generator) to extract the content from the web pages stored in a DSPR. The Extraction Pattern Generator takes two inputs: the labels from the search forms as input and the keywords of the cluster matching the label on the search form. The six types of the extraction patterns shown in Figure 6 have been used by the EP Generator in this work.

Figure 5. System for domain identification and page classification

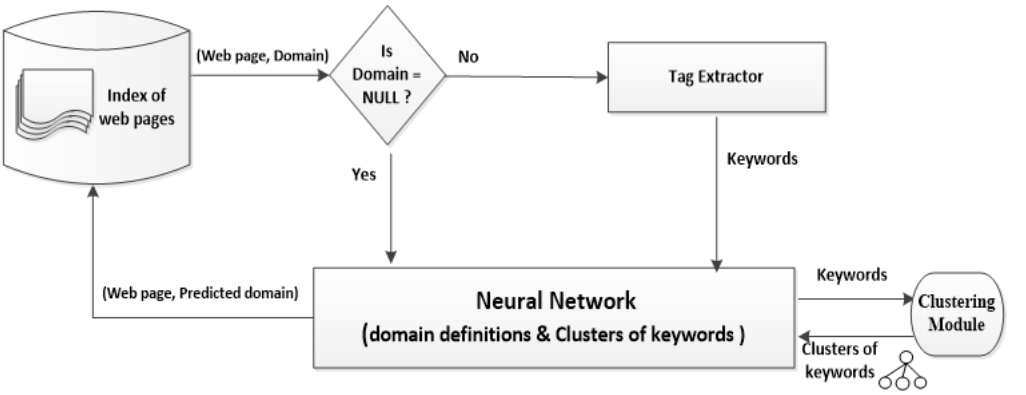


Table 1. A sample domain-specific database

Attributes	Values
Source; Departure City; From	Delhi; Mumbai
Destination; Arrival City; To	Delhi; Mumbai, Chennai

Figure 6. Extraction patterns

Extraction patterns:

EP1: Ls like NP1, ..., NPn

EP2: Ls for example NP1, ..., NPn

EP3: Ls such as NP1, ..., NPn

EP4: Ls comprise of NP1, ..., NPn

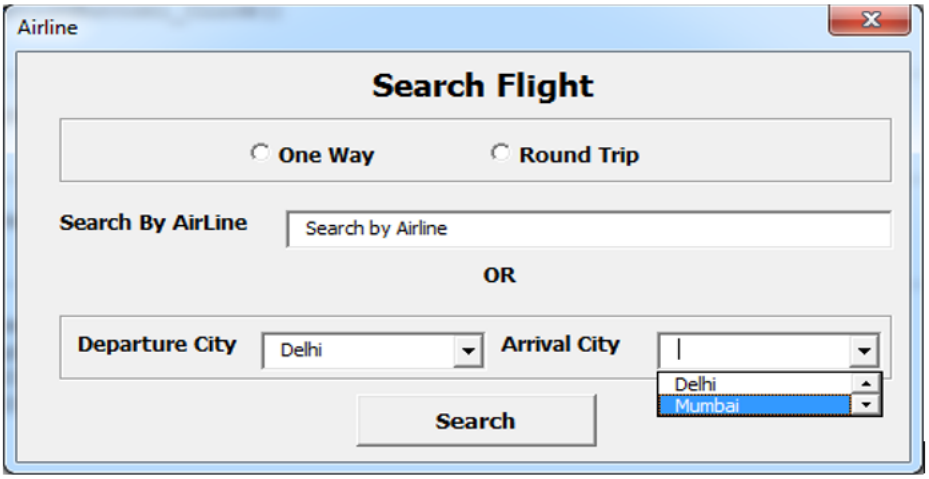
EP5: Ls including NP1, ..., NPn,

EP6: Ls in contrast to NP1,...NPn

Where *EPi* is any extraction pattern.

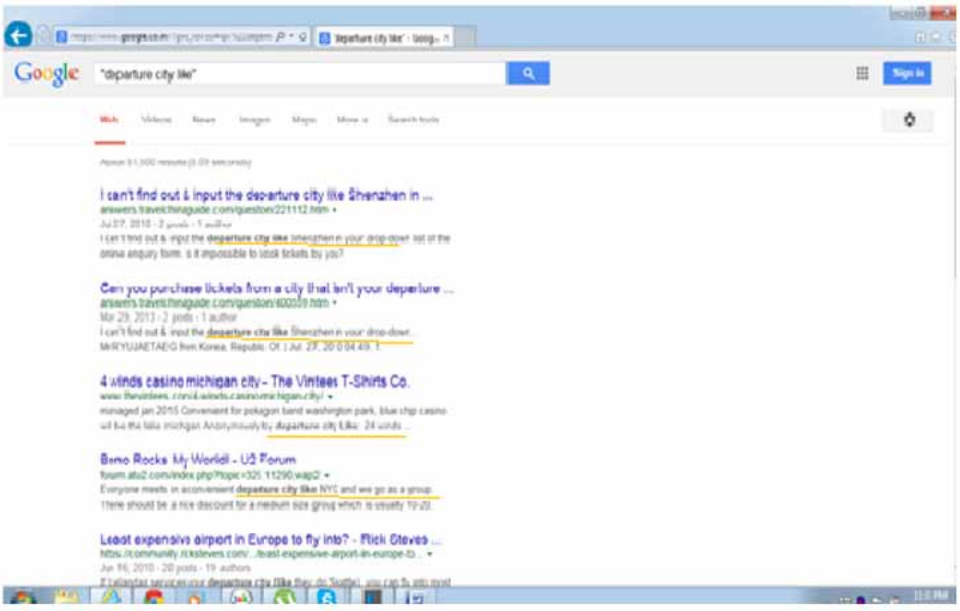
Here, Ls indicate the attribute/label like Source, Destination etc., from search forms and NP1, ..., NPn indicate the values / instances that can be taken by that attribute/ label in the pattern. For example, consider the sample form from 'Travel' domain as shown in Figure 7. The search form consists of various control elements: a textbox control labelled Search by Airline that takes a free-form input, a radio button labelled Search Flight giving the option for selecting a One-Way travel or a Round Trip and two combo boxes labelled Departure City& Arrival City for selecting the departure and arrival cities of travel respectively.

Figure 7. A sample search form



The labels are extracted from the search form with the help of the Form Analyser and Classifier (FAC). The working of FAC is explained in the later section. The labels are then used to create the extraction patterns which are then raised as queries on the webpages in the DSPR. For example, by using the label 'Departure city' extracted from the above search form, such as "departure city like", "departure city for example", "departure city such as" etc. can easily be formed. Figure 8 shows a webpage from the DSPR for Travel domain. The web page is scanned for the occurrences of the extraction pattern, say "Departure city like".

Figure 8. Google web page for the query "departure city like"



As can be observed, the specified pattern occurs at many locations in the web page (marked yellow) giving instances like:

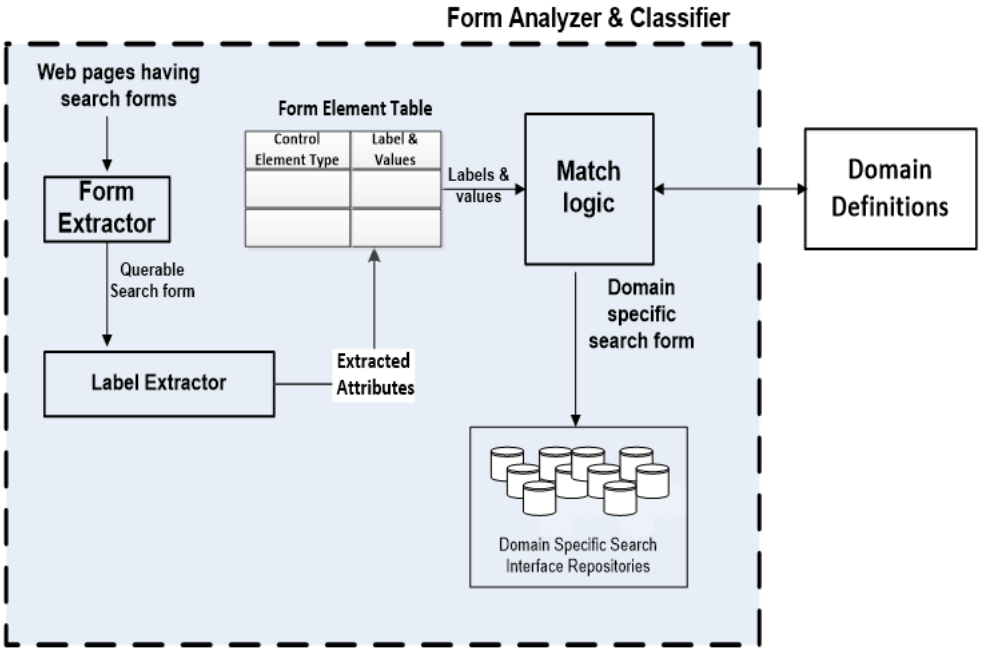
- Departure city like Shenzhen.
- Departure city like NYC.
- Departure city like Tokyo, Kyoto, Hiroshima and Osaka.
- Departure city like Rome, London, Frankfurt, Milan etc.
- Departure city like Amsterdam heading through Germany to Switzerland.

The retrieved instances are then tokenized to find the values for different attributes or labels. The following values are retrieved for the label 'departure city': Shenzhen, NYC, Tokyo, Kyoto, Hiroshima, Osaka, Rome, London, Frankfurt, Milan, Amsterdam, Germany and Switzerland.

Form Analyser and Classifier

The Form Analyser and Classifier (FAC) takes the web page containing the search forms from the Page Parser and Distributor as the input and creates the Domain-Specific Search Interface Repository containing the search forms as the outcome. Each Domain-Specific Search Interface Repository contains search forms for a specified domain. For example Domain-Specific Search Interface Repository for Books domain contains the search forms for only Books domain. The working of the Form Analyser and Classifier (FAC) can be explained with the help of Figure 9.

Figure 9. The Form Analyser and Classifier



The FAC extracts the search form from the Form Pages and checks if it is a query able form or registration form. A query able form is a form that upon filling with valid values returns a web page that has been dynamically generated from the web data-base. It has been observed that Query able

forms are typically designed to offer search functionality and so have the submit buttons which are typically found to be named as "GO" or "Search" whereas the forms that have buttons named as "SIGN IN" or "LOGIN" force prior registration of the user before carrying any search.

A Form-Element Table (FET) is then generated for each query able form by extracting the labels of the various control elements, the type of the control elements and values that are associated with any control element. For example, for the search form in Figure 9 the corresponding FET that is generated is shown in Table 2.

Table 2. Parsed representation for the above form as Form Element Table (FET)

Control Element (E)	Label (L)	Values/Dom(E)
Select	Departure City	Delhi, Mumbai
Select	Arrival City	Delhi, Mumbai
Radio	Flight trip	One-way, Round trip
Text	Search by airline	String of characters
Submit	Search	Submit

Some control elements like the ones labelled as Departure City, Arrival City, and flight trip offer a finite list of possible values such as select-option, checkboxes or radio buttons which are embedded in the webpage itself. Such values are extracted for inclusion in the FET. In general, if E is any control element, then Dom (E) is the set of values that are valid as input to E. For example: for label 'Arrival City' and element type Select the Dom (E) = {Delhi, Mumbai}.

The most important task of Form Analyser & Classifier is to find the domain (Auto, Books, Food, and Travel) of the candidate search form so as to discover the Hid-den Web resources relevant in any domain. This is done with the help of the Match Logic that makes uses of the Domain Definitions generated during the classification of PIWPs by the Page Analyser and Distributor. The Match Logic helps to semantically map the search form with the Domain Definitions. The Match Logic supports multi-strategy matching i.e. uses multiple strategies like Edit Distance Algorithm, Domain-Specific Thesaurus etc. for matching the two inputs.

The Match Value Generator associates a Boolean variable termed 'MATCH' with each label. The Match logic sets the value of the variable 'MATCH' to TRUE if the label matches with a key term in some Domain Definition whereas a "MATCH" value of FALSE for the label indicates that there does not exist any match for that label in any of the Domain Definitions. Besides indicating a true/false match, the Match Logic also associates the domain of the matched key term with that label. This will ease the task of identifying the domain of the search form. If the number of labels of a FET that match with the key terms from a common Domain Definition are more than a set cut-off, then it is qualified as a search form in that domain.

As an example, consider the search form in Figure 10. In this scenario when the Match logic generates the matches between the labels of the FET and the key terms of the various domain definitions, the "MATCH" field is always assigned a TRUE value. Moreover, all the labels get associated with the key terms of a common do-main definition of Books as shown in the 4th column of the Table 3.

All seven labels match to the key terms in the definition of Books domain i.e. the search form exhibits a 100% match with the Books domain, thereby predicting the Hidden Web resource as relevant in Books domain. Although, the label price also adopts definition from auto & Travel domains other than that of Books, the match percentage is less than 15% (only 1 out of the 7 label match) and hence

Figure 10. A sample search form from Books domain

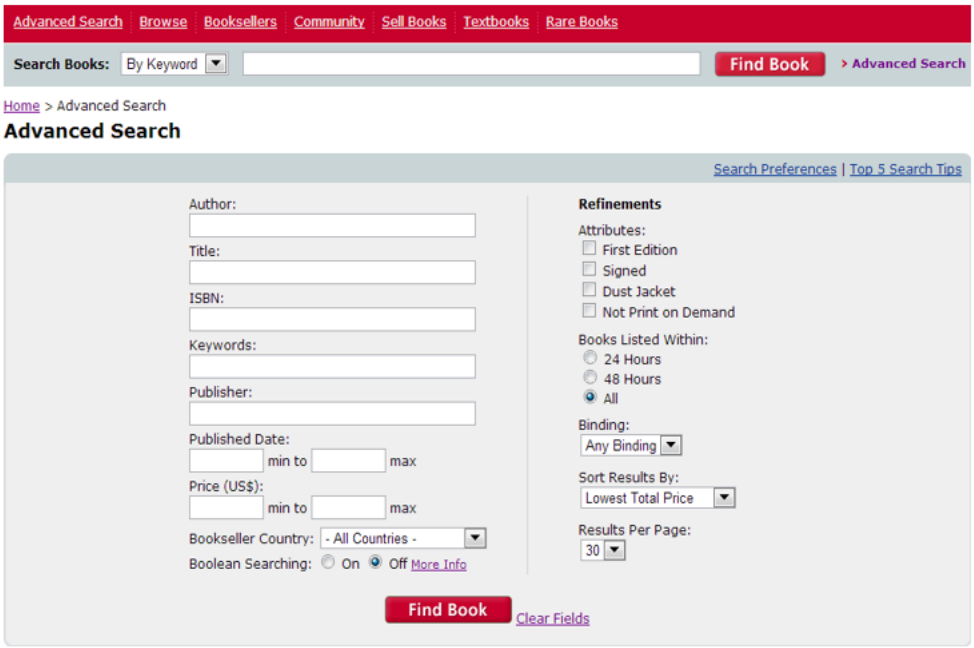


Table 3. Labels and type of input associated with them for the search interface form in Figure 10

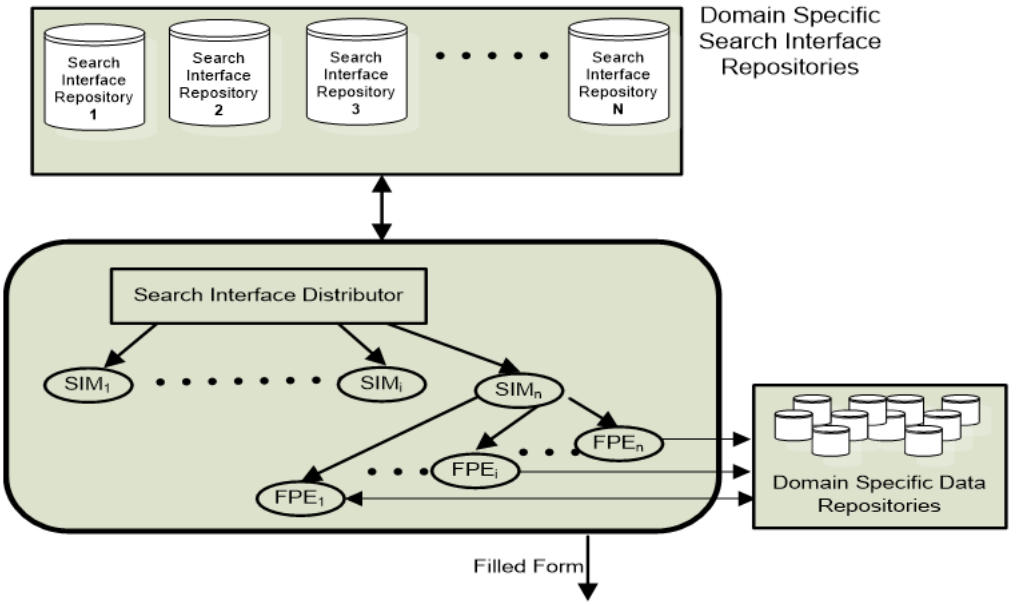
Extracted Labels on the search Interface	Control Type	Match	Domain of the Mapped Attribute
Author	Text	TRUE	Books
Book	Text	TRUE	Books
ISBN	Text	TRUE	Books
Price	Text	TRUE	Books, Travel
Publisher	Text	TRUE	Books
Title	Text	TRUE	Books
Find Books	Text	TRUE	Books

the resource does not qualify for relevance in any other domain. The various hidden web re-sources after examination have been stored in the various Domain-Specific Search Interface repositories. Also, these repositories are open for updating whenever any new relevant hidden web resource or a search form is discovered in a domain.

Search Interface Distributor

This component of the crawler takes the various Domain-Specific Search Interface Repositories as input and processes the search forms in parallel to efficiently retrieve the contents from the Hidden Web as illustrated in Figure 11.

Figure 11. Processing the search forms in parallel



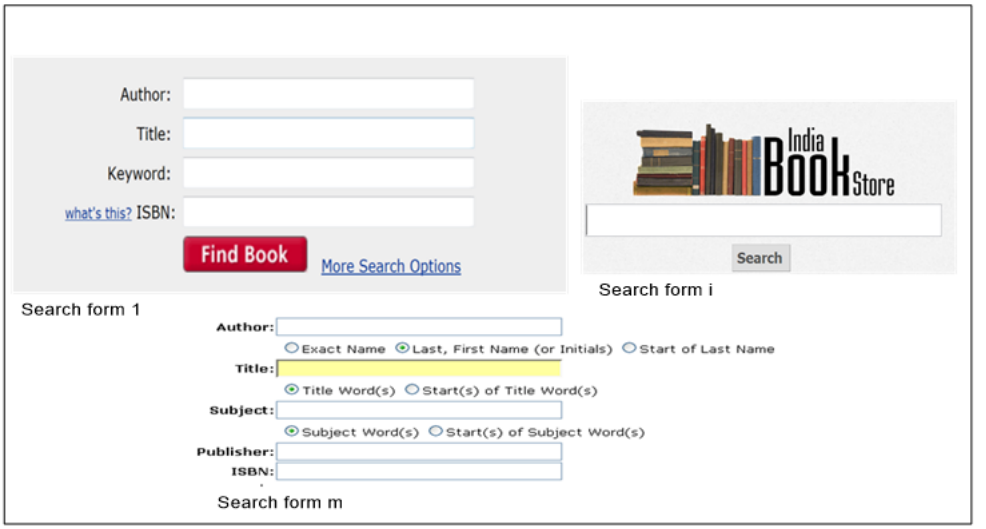
In order to exploit parallelism, the crawler should be able to process and submit the search forms in parallel. Therefore, in the proposed architecture parallelism has been incorporated in following two levels:

1. At the level of Search Interface Managers (SIM): The Search Interface Distributor (SID) acts as the coordinator at this level. It is responsible for distributing the search interface repositories among the different search interface managers or SIMs according to their domains. For example, the search interface repository having numerous search forms from the Books domain is assigned to the search interface manager that is held responsible for processing the forms from the Books domain. Similarly, all other SIMs are assigned the responsibility for pro-cessing the search forms that are contained in the search interface repository respective to their assigned domain. Distributing the search forms domain-wise helps in avoiding overlap and redundancy.
2. At the level of Form Processing Elements (FPE): At this level, SIM, that is containing a repository of search forms for a particular domain, distributes the search forms to every form processing element (FPE) is responsible to fill that form in future. For example, consider the sample of the search form repository from Books domain shown in Figure 12 that contains only those search forms which al-low online searching of books.

The SIM is responsible for fetching the hidden web data from the Books domain and distribute these m forms to m Form Processing Elements (FPE). This enables parallel harvesting of hidden web data in the Books domain through the parallel pro-cessing of search forms by the various Form Processing Elements created by the SIM responsible for it.

Now, as each SIM is assigned only the search forms from a fixed, unique domain, the number of SIMs is constant in a single crawl and equals the number of distinct domains of crawler consideration. This guarantees that all the search interface repositories have been harvested. But as all the search interface repositories are initially mapped to the set of SIMs, the crawler cannot increase the number of Search Inter-face Managers to accelerate processing of search forms (the crawling process). Therefore,

Figure 12. Sample of the Domain-Specific search interface repository in Books domain



to overcome this limitation on the crawl rate, each SIMs is made responsible for a random distribution of its load among the parallel Form Processing Elements (FPE) to facilitate load balancing.

Also, all the FPEs of a Search Interface Manager need to register themselves with their representative SIM at that instant before a request for processing the list of search form is made by it. Each FPE can be assigned a maximum of n search forms for processing by it. If the search interface repository of a SIM contains more than n forms, then the SIM creates a new instance of the FPE and assigns another n forms from its repository to the newly created FPE. This assures an even distribution of workload among all the form processing elements that were registered with a particular SIM. Thus, the number of FPEs must be created and destroyed dynamically by the respective SIM as per the requirement in its domain. The Load distributor function used by the SIM is shown in Figure 13.

After distribution, each form processing element is held responsible for harvesting exclusively the set of search forms received from its SIM without communicating directly with each other. Whenever a FPE starts, it is initialized with the parsed representation (generated by FAC) of the search form to be processed, con-tent filter, list of URLs to be included and excluded from crawling etc.

A FPE analyses the various parameters to judge the type of the contents that is required to fill up the search form. The Domain-Specific Databases act as filling resources for the form processing elements. The Form Processing Element generates ranking among the queries listed in the Domain-Specific database based on the statistics that is collected by the crawler during its execution. The detailed ranking mechanism used by the FPE is provided in (Gupta & Bhatia, 2014).

The FPE then fills the form by raising queries as per their ranks and associating a suitable value with each control, the value being chosen from the domain of the respective control element. The working of an individual form processing element can be explained with the help of the algorithm shown in Figure 14. The FPE, after filling the search form, submits the request to the corresponding Web Server and obtains valuable responses in the form of dynamically generated web pages that form the crawled repository of Hidden Web Pages. The obtained set of response pages are further analysed to collect data for the different repositories mentioned in the architecture.

Once the FPE is over with the set of search form assigned for crawling, it makes a request for new search forms to be assigned to it. The Search Interface Manager does not need to permanently monitor the system because the FPEs demand work on a need basis. The FPEs does not impose any

Figure 13. Algorithm to distribute load by the SIMs to the associated FPEs

```
Load_ distributor()

Input:  A domain-specific search interface repository {s1, s2,s3.....sz}
Output: workload of each FPE

1.      For each search form in the repository
2.        if (z≤n) then
                {
3.              create a FPE
4.              assign the forms s1 to sz  to the  FPE;
                }
5.        else
                {
6.               m= u_bound(z/n)
                //u_bound() returns an upper bound for the value
7.               create m FPE
8.              assign n forms sequentially from the repository to each of m FPE;
                }
9.    end for
```

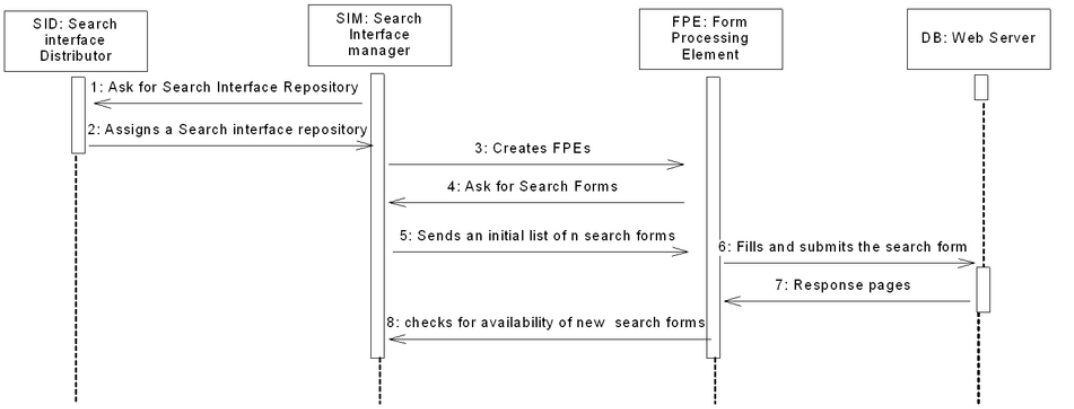
Figure 14. Algorithm of a Form Processing Element (FPE)

```
Form_Processing_Element()

Input:  A list of search interfaces in a domain {s1, s2 , s3.....sn}
Output: response page

1.      For the FET of each search form interface si in the assigned list
        {
2.              Match (control labels, attributes in domain specific database)
3.              Associate (control labels, matched attributes);
4.              Assign values to labels based on the order given by the Query Ranker;
5.              Submit the filled search interface ;
6.              Download the response page;
8.              Forward the response page to the Response Analyzer;
9.      }
```

overhead on the Search Interface Manager because the SIM simply responds to requests for unprocessed search forms and all the SIMs and the FPEs work in a recursive fashion till the specified time or the exhaustion of other available resources.

The interaction between the SID, SIM, FPE and the hidden web database can be illustrated with the help of a sequence diagram as shown in Figure 15.

Figure 15. A sequence diagram showing interaction between the SID, SIM, FPEs and the Web server



The sequence diagram consists of four objects, one each of a SID, SIM, FPE and the database server. It consists of the following steps:

1. The SIM created for a domain requests the SID for the Search Interface Repository associated with the domain of its consideration.
2. The SID provides the corresponding Search Interface Repository to the SIM.
3. The SIM creates a FPE for processing the provided Search Interface Repository.
4. The FPE then asks its controlling SIM for the n search forms to be processed by it.
5. The SIM sends the requested number of search forms to the FPE.
6. The FPE processes each search form by filling with appropriate values and submits the filled search forms to the Web Server.
7. The Web Server generates dynamic web pages in response which are then passed to the Page Parser and Distributor component of the proposed crawler.
8. After the FPE has processed the n search forms assigned to it, it asks the con-trolling SIM for more search forms if available in the Search Interface Repository to continue harvesting the Hidden Web data.

The SID object remains active for steps 1 and 2; the SIM object remains active for steps 1,2,3,4,5,8; the FPE object remains active for 3,4,5,6,7,8 and the Web Server remains activated during steps 6 and 7.

The distribution of the search forms from the Search Interface Repository as-signed to the SIM is done dynamically during the crawl by the SIM itself through its support to variable number of form processing elements.

The design of this framework helps to combat the problem of:

1. **Overlap:** For the proposed hidden web crawler, overlap is said to exist if and only if the same search form is considered for processing by more than one SIM as this will lead to multiple fetches of the same webpage and thus a redundancy in the extracted hidden web content. However, the proposed approach avoids overlap by assigning the search form to a unique SIM respective to its domain.
2. **Synchronization:** The Search Forms have to be partitioned to enable parallel crawling but this involves an overhead of synchronizing the parallel crawling processes to minimize overlap. The proposed approach does not involve the overhead of synchronizing the parallel threads as the Search Interface Distributor acts as the centralized coordinator of the operation of all these parallel threads or elements and distributes the work as per the do-mains.
3. **Communication Bandwidth:** Although, each SIM need to coordinate with its associated FPEs but neither the various SIMs nor the FPEs need to communicate among themselves to coordinate with each other. This helps to minimize the communication bandwidth needed for synchronizing the work of the various parallel threads. Moreover, avoiding the overlap among the search forms during processing will be useful to eliminate repeated har-vesting of the same hidden web database which further helps the proposed Parallel Hidden Web Crawler to preserve the network bandwidth and thus improve the efficiency and the effectiveness of the crawler.

The details of implementation and the results of the various experiments that were conducted to evaluate the proposed work are discussed in the next section.

EXPERIMENTAL RESULTS AND IMPLEMENTATION

The proposed Parallel Hidden Web Crawler has been implemented using .NET framework but it also includes software components implemented in native code i.e. in other programming languages, such as C, C++ but can be inter-operated with java code using the JAVA Native Interface (JNI).

Consider C as the number of valid or correct search form submissions; I, the number of invalid or incorrect search form submissions and N, the number of search forms that could not be filled and submitted by the proposed Parallel hidden web crawler. For the purpose of analysing the performance, the following metrics have been used:

Precision is defined as a fraction of correct form submissions over all the form submissions by the Parallel Hidden Web Crawler. Mathematically, the Precision is given by:

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

Recall is defined as a fraction of correct form submissions over all the search forms given to the system for processing by the crawler, then the Recall of the proposed parallel hidden web crawler crawling system is given by the expression given in equation 2:

$$R = C/ (C + I + N) \quad (2)$$

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.

Figure 16. Initial list of URLs for 'Food' domain

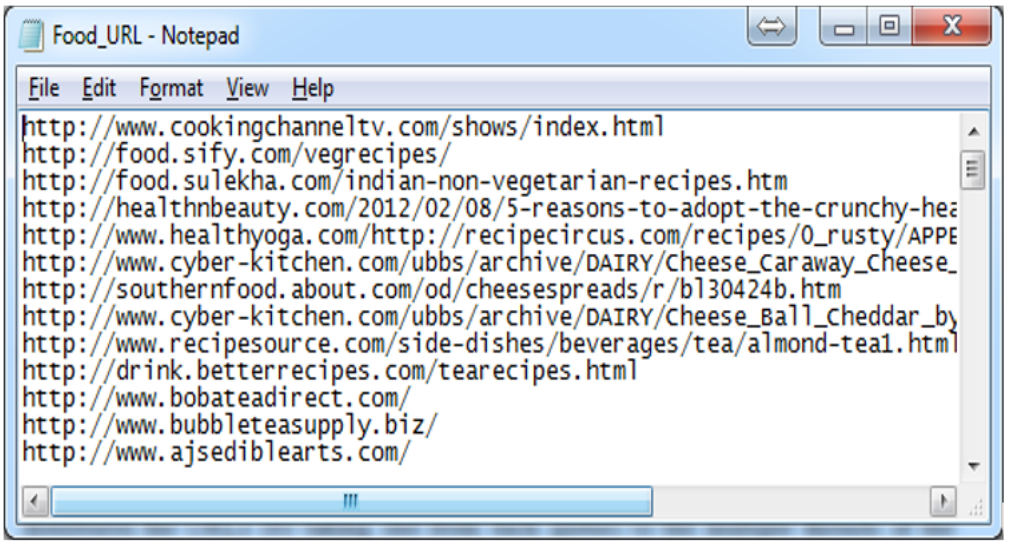
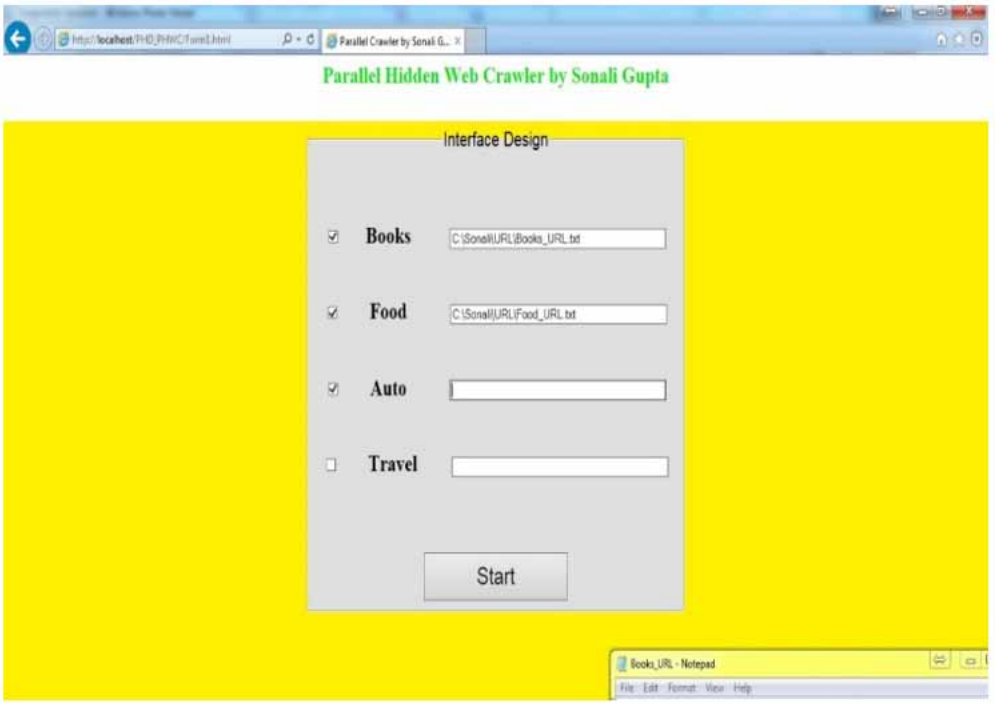


Figure 17. Interface of the Parallel Hidden Web Crawler



Data Sets

For experimental evaluation of the proposed work, the following four domains have been considered: Auto, Books, Food and Travel. The proposed crawler is initialized by taking about 140-150 URLs for each domain from the list provided under the DMOZ open directory that were stored in the various Domain-Specific URL Queues. A sample of the initial URL queue for the Food domain contained in the text file is as shown in Figure 16.

Similarly, URLs were taken for other domains and arranged in order in the respective URL Queues (.txt files). These text files or queues were given as inputs in each domain as shown in the interface of the crawler in Figure 17.

Results

A total of 565 pages were downloaded by the Multi-threaded Document Loader and were given to the Page Parser and Distributor that classified them as per Table 4.

Each of the PIWP and the Form page, immediately after analysis is passed respectively to the Page Analyser and Classifier (PAC) and the Form Analyser and Classifier (FAC). Of the total 383 search forms received by FAC, 369 were found to be query able. When these 369 forms were analysed, 322 were discovered as relevant search forms in one or the other domain but could not predict the relevant domain of 27 search forms. The observed data is shown in Table 5.

Table 4. The pages of each type as identified by the page type identifier

Total Pages downloaded	PIW Pages	Form Pages
565	182	383

Table 5. Results of the Form Analyser

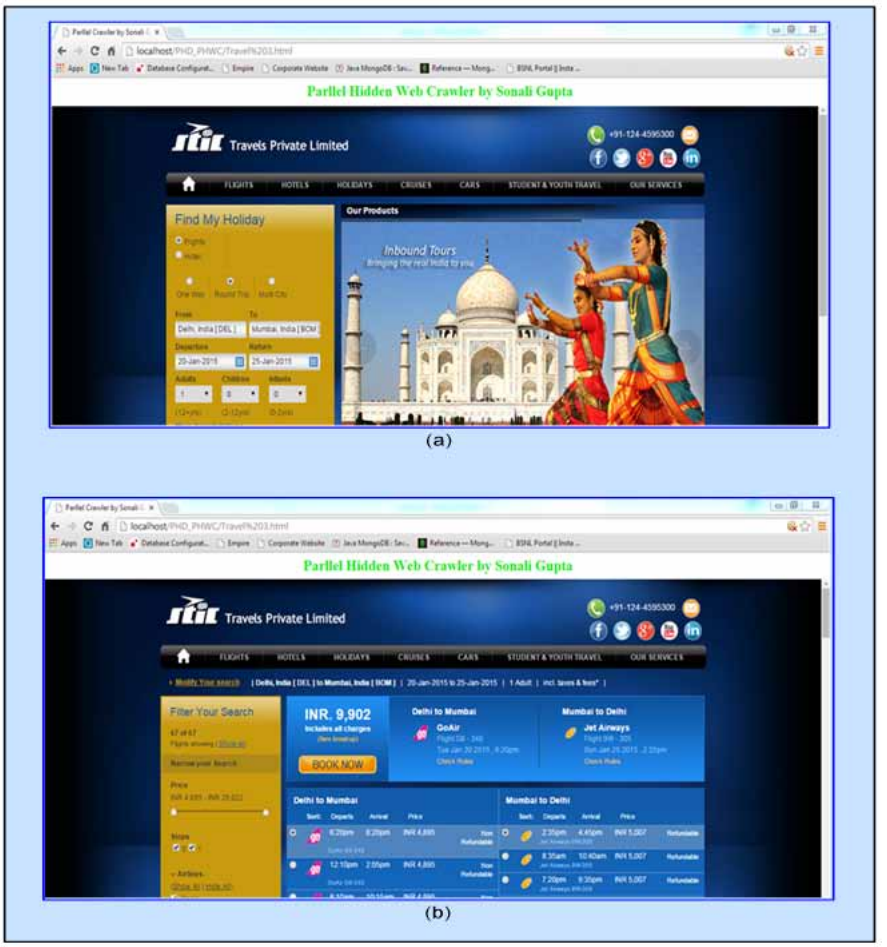
Total # of search forms in the Hidden Web	369
# correctly discovered relevant search forms	299
# incorrectly discovered search forms	43
# search forms that could not be classified	27

After these various Domain-Specific Databases and the search interface repositories were created, the search forms are processed in parallel within and among domains.

The top ranked query in the Domain-Specific Database (as suggested by the ranking mechanism used by FPE) is used to fill in all the search forms respective to that domain. Figure 18(a) shows a search form from Travel domain that is processed by the query From= Delhi, To= Mumbai and Search Flight= Round-trip. Figure 18(b) shows the corresponding dynamic web page retrieved in response of this form sub-mission.

Figure 19 and 20 shows a snapshot of the crawler's task of filling the search forms in parallel along different domains. Figure 19 shows that all the forms from the Travel domain when processed in parallel, are filled using the same query (flights making a round-trip from Delhi to Mumbai on the specified dates) and Figure 20 shows the response pages that were downloaded when the corresponding search forms were processed.

Figure 18. Sample search form from travel domain that is processed and the corresponding response page retrieved



In the same way, the search forms from all the other three domains (Auto, Books and Food) are processed with the help of the FPEs.

COMPARISON OF ALL DOMAINS

It may be observed from the data in the Table 6 that the number of search forms that could not be processed by the system (which was 14+13+8+11=46) is approximately the same as the number of hidden web resources that were assigned to a domain irrelevant for the search form by the FAC. Thus, an incorrect domain-specific data repository based on the predicted domain of the hidden web resource (which is in fact different than the actual domain of the search form), might have been used to process the search form, making the crawler incapable of processing it.

Based on the data in Table 6, the precision, recall and f-measure values are computed in each of the domains for the proposed crawler.

It may be observed from the graph in Figure 21 that on an average while pro-cessing search forms in any domain, the Precision is high i.e. ranges from 75.21% to 89.18%, range of Recall is also high i.e. from 76.66% to 91.2% and average F-measure of is also quite high i.e. varies from 75.12% to 90.03%.

Figure 19. Parallel Processing of search forms in Travel domain with the query round-trip flights from Delhi to Mumbai on specified dates

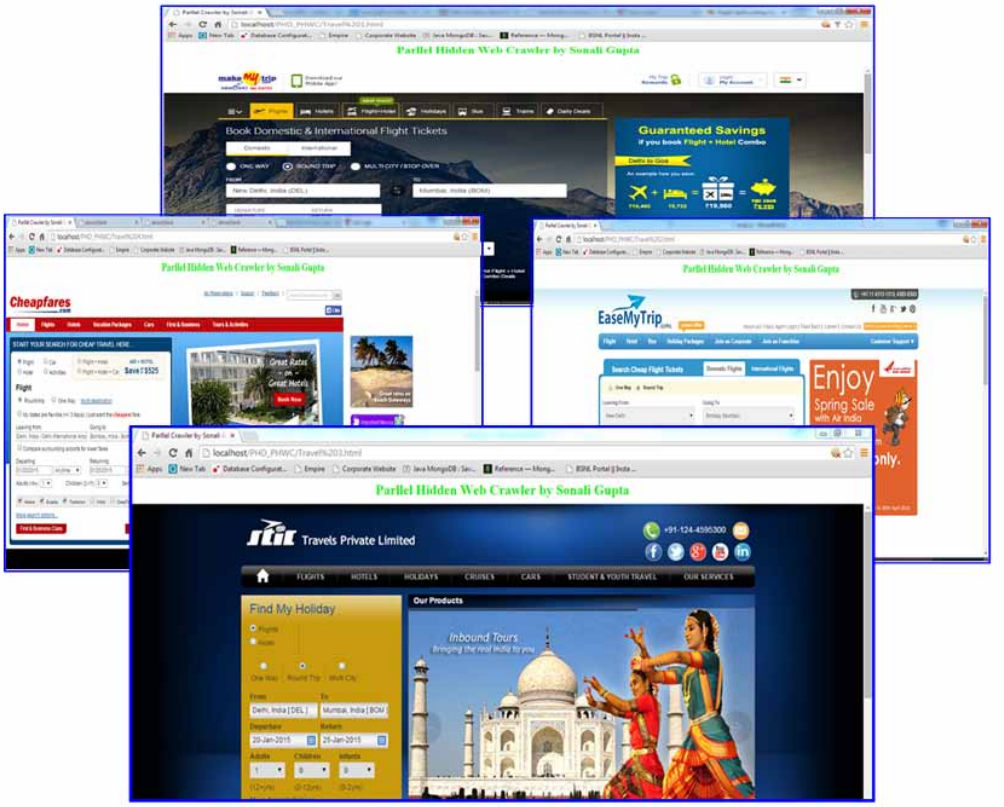


Figure 20. Parallel downloading of response pages from the Hidden web databases behind the search forms in Figure 19

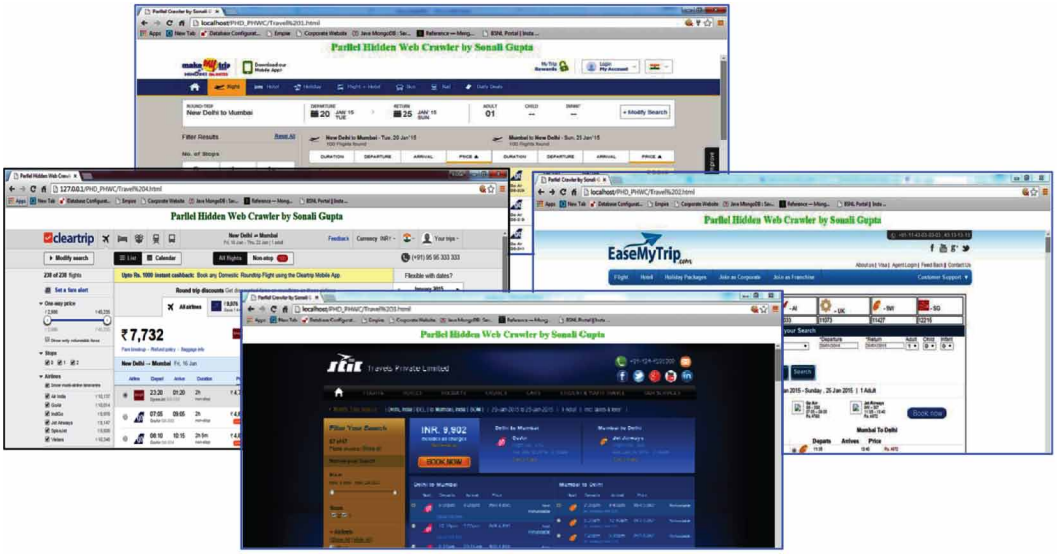
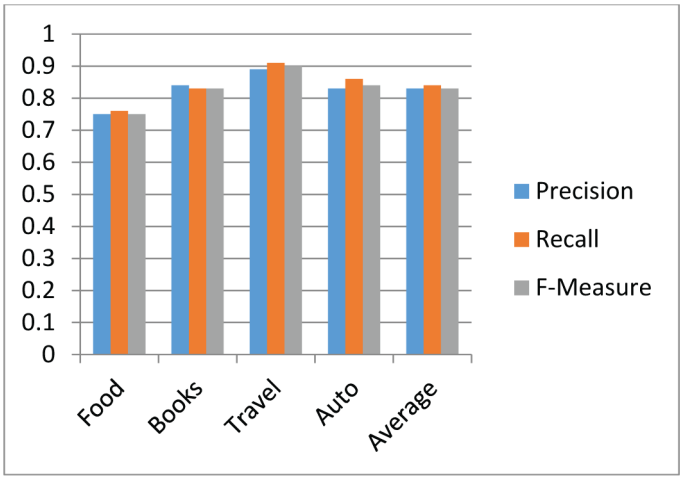


Table 6. Number of form submissions: Valid, Invalid and Failures

Domain of search forms	Average no. of Forms successfully submitted and processed		Average no. of failures in form submission and processing, N
	Valid form submissions C	Invalid form Submissions I	
Food	44	15	14
Books	62	12	13
Travel	79	10	8
Auto	65	13	11

Figure 21. Precision, Recall and F-Measure values in each domain and average over all domains



It may be noted that the maximum value of precision, recall and f-measure is observed for the web search forms in travel domain. The reason behind this is the performance of the Page Classifier and the Content Extractor that helps in creating a domain-specific database that is highly accurate.

A slightly low value of Precision, Recall and F-measure is observed in the Food domain occur due to the high number of cuisines that exist based on the geography of the region. Moreover, the same ingredient used for different food items in cooking has been assigned different names depending on the language used by the people in that region. However, during the experiment, it was found that the values of Precision, Recall and F-measure for the Books, Auto and Travel domains remain fairly consistent. Thus, the experiments conducted over all the domains indicate that the framework proposed in this work is both consistent and efficient.

CONCLUSION

An effective and efficient technique to crawl and extract the content in the Hidden web databases has been designed and implemented in this work. More specifically, the main challenges involved in developing a parallel crawler that downloads pages from the Hidden Web (Barbosa & Freire, 2004; Wang et al., 2010) have been addressed and resolved. To tackle the issue of scale of the Hidden Web, a parallel approach that provides effective coverage by following a domain-specific approach and efficiently crawling the huge contents in the Hidden web has been proposed and implemented.

As the size of the hidden web contents are very large and it would continue to grow with the time. Therefore, work can be done to design a search engine that could be able to crawl, extract and index the content of these hidden databases.

REFERENCES

Barbosa, L., & Freire, J. (2004). Siphoning hidden-web data through keyword-based interfaces. SBBD, 309-321.

Bergholz, A., & Chidlovskii, B. (2003). Crawling for domain-specific Hidden Web resources. In *Proceedings of the Fourth International Conference on Web Information Systems Engineering (WISE'03)*. IEEE Press. doi:10.1109/WISE.2003.1254476

Chakrabarti, S., van den Berg, M., & Dom, B. (1999). Focused crawling: A new approach to topic-specific web resource discovery. *Computer Networks*, *31*(11-16), 1623–1640. doi:10.1016/S1389-1286(99)00052-3

Cheng, S., Termehchy, A., & Hristidis, V. (2012). Predicting the Effectiveness of Keyword queries on Databases. *ACM Conference on Information and knowledge Management (CIKM)*.

Cho & Garcia-Molina. (2002). *Parallel Crawlers*. Academic Press.

Diligenti, M., Coetzee, F., & Lawrence, S. (2000). Focused crawling using context graphs. *Proc. of 26th International Conference on Very Large Data Bases*, 527–534.

Gravano, L., Ipeirotis, P. G., & Sahami, M. (2003). QProber: A System for Automatic Classification of Hidden-Web Databases. *ACM Transactions on Information Systems*, *15*(1), 1–41. doi:10.1145/635484.635485

Gupta & Bhatia. (2014). Optimal Query Generation for Hidden Web Extraction through Response Analysis. *International Journal of Information Retrieval Research, 4*(2), 1-18.

Gupta, S., & Bhatia, K. K. (2012). Exploring Hidden parts of the Web: The Hidden Web. *Proceedings of the International Conference ArtCom*, 508-515. doi:10.1049/cp.2012.2556

Gupta, S., & Bhatia, K. K. (2013a). Domain Identification and Classification of Web Pages Using Artificial Neural Network. In *Advances in Computing, Communication, and Control. ICAC3 2013. Communications in Computer and Information Science* (Vol. 361). Springer. doi:10.1007/978-3-642-36321-4_20

Gupta, S., & Bhatia, K. K. (2013b). HiCrawl: A Hidden Web crawler for Medical Domain. *Proceedings of 2013 IEEE International Symposium on Computing and Business Intelligence,* 152-157.

Kashyap, A., Hristidis, V., Petropoulos, M., & Tavoulari, S. (2011). Effective navigation of Query results based on Concept Hierarchies. *IEEE Trans. Knowl. Data Eng.*, *23*(4), 540–553. doi:10.1109/TKDE.2010.135

Lawrence, S., & Giles, C. (1998). Searching the World Wide Web. *Science*, *280*(5360), 98–100. www.sciencemag.org. doi:10.1126/science.280.5360.98 PMID:9525866

Liu, J., Wu, Z., Jiang, L., Zheng, Q. H., & Liu, X. (2009). Crawling Deep Web Content Through Query Forms. *Proceedings of WEBIST 2009*, 634-642.

Madhavan, Ko, Kot, Ganapathy, Rasmussen, & Halevy. (2008). Google's Deep-Web Crawl. *Proceedings of Very large data bases*, 1241-1252.

Ntoulas, A., Zerfos, P., & Cho, J. (2005). Downloading Textual Hidden Web Content Through Keyword Queries. *5th ACM/IEEE Joint Conference on Digital Libraries JCDL05*, 100-109. doi:10.1145/1065385.1065407

Qi & Brian. (2007). *Davison: Web Page Classification: Features and Algorithms*. Department of Computer Science & Engineering, Lehigh University.

Qi, X., & Davison, B. D. (2006). Knowing a web page by the company it keeps. *International conference on Information and knowledge management (CIKM)*, 228-237. doi:10.1145/1183614.1183650

Sharma, A. K. (2008, December). A Framework for Domain-Specific Interface Mapper (DSIM). *International Journal of Computer Science and Network Security*, *8*(12).

Shibu, Vishwakarma, & Bhargava. (2010). A combination approach for Web Page Classification using Page Rank and Feature Selection Technique. *International Journal of Computer Theory and Engineering, 2*(6).

Wang & Lochovsky. (2003). Data Extraction and Label Assignment for Web Databases. In WWW 2003. ACM.

Wang, Lu, Liang, Chen, & Liu. (2010). Selecting queries from sample to crawl Deep Web Data Sources. *Web Intelligence and Agent Systems, an International Journal*.

Xiang, P., Ke, T., & Huang, Q. (2008). A Framework of Deep Web Crawler. *Proceedings of the 27th Chinese Control Conference*.

Sonali Gupta is working as Assistant professor in the department of Computer Engineering, Faculty of Informatics and Computing, JC Bose University, YMCA Faridabad since last 10 years and has more than 16 years of experience in teaching and Research.