A Framework for Incremental Hidden Web Crawler

Rosy Madaan
Computer Science & Engineering
B.S.A. Institute of Technology & Management
Faridabad, India

Ashutosh Dixit
Department of Computer Engineering
Y.M.C.A. University of Science &Technology
Faridabad, India

A.K. Sharma
Department of Computer Engineering
Y.M.C.A. University of Science &Technology
Faridabad, India

Komal Kumar Bhatia
Department of Computer Engineering
Y.M.C.A. University of Science &Technology
Faridabad, India

***Abstract*—Hidden Web's broad and relevant coverage of dynamic and high quality contents coupled with the high change frequency of web pages poses a challenge for maintaining and fetching up-to-date information. For the purpose, it is required to verify whether a web page has been changed or not, which is another challenge. Therefore, a mechanism needs to be introduced for adjusting the time period between two successive revisits based on probability of updation of the web page. In this paper, architecture is being proposed that introduces a technique to continuously update/refresh the Hidden Web repository.**

Keywords-WWW; Surface Web; Hidden Web; Search engine; Crawler

I. INTRODUCTION

WWW [1, 12, 16] can be broadly divided into two parts: Surface Web and Hidden Web [2, 6, 10]. The Surface Web refers to the part of the Web that can be crawled and indexed by general purpose search engines [3], while the hidden Web refers to the abundant information that is "hidden" behind the query interfaces and not directly accessible to the search engines. Hence, there is need to access the Hidden Web through their query interfaces. UUIC survey indicates that there are more than 300,000 Hidden Web databases [2,6,10] and 450,000 query interfaces available on the Web. The contents on Hidden Web are not only increasing but also spanning well across all topics.

The main characteristics of the Hidden Web are as follows:

- It has Broad, relevant Coverage,
- It contains High quality contents, and
- Its contents exceed all printed contents.

The existing web crawlers can retrieve only Surface web pages ignoring the large amounts of high quality information 'hidden' behind search forms that need to be filled manually by the user. A search interface consists of many forms elements like textboxes ,labels ,buttons etc. and the user is expected to provide data in at least one of them before submitting the form in order to obtain the response pages containing the results of the query.

In order to download the Hidden Web contents from the WWW the crawler needs a mechanism for Search Interface interaction i.e. it should be able to download the search interfaces in order to automatically fill them and submit them to get the Hidden Web pages as shown in Fig. 1.

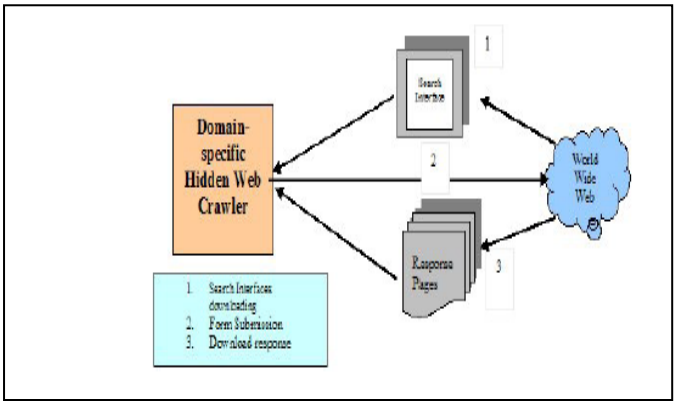

Figure 1. Crawler Search Interface Interaction

The retrieved Hidden web documents are thereof stored in a repository. The Indexing function is performed by the Indexer [3] module of the Search engine. User provides a query on the query interface of the Search engine, the index is then searched for finding out a corresponding match, if any and the results are returned to the user.

In this paper, a framework has been proposed that updates the repository of search engine by re-crawling the web pages that are updated more frequently. The paper has been organized as follows: section 2 describes the current research that has been carried out in this area; section 3 describes the proposed work to crawl the hidden web documents incrementally; section 4 shows the performance of proposed work and last section concludes the proposed work.

II. RELATED WORK

The notion of a Hidden (or Deep or Invisible) Web has been a subject of interest for a number of years. A number of researchers have discussed the problems of crawling the contents of hidden Web databases [3, 8, 12, 9, 10, 11 12, 14, 16] as follows.

Raghavan and Garcia-Molina [14] proposed HiWE, a task-specific hidden-Web crawler, the main focus of this work was to learn Hidden-Web query interfaces. Lin and Chen's [10] built a catalogue of small search engines located in sites and choose which ones were more likely to answer the query. However, forms with more than a text field was not treated. Wang and Lochovsky [9] described a system called, DeLa, which reconstructed (part of) a "hidden" back-end web database, and it used the HiWE. There are other approaches that focused on the data extraction. Lage et al. [12] claimed to automatically generate agents to collect hidden web pages by filling HTML forms. Liddle et al [12] performed a study on how Valuable information could be obtained behind web forms, but did not include a crawler to fetch them. Barbosa and Freire [11] experimentally evaluated methods for building multi-keyword queries that could return a large fraction of a document collection.

Xiang Peisu et al. [16] proposed model of forms and form filling process that concisely captures the actions that the crawler must perform to successfully extract Hidden Web contents. It described the architecture of the deep web crawler and described various strategies for building (domain, list of values) pairs. Although this work extracts some part of Hidden Web but it is neither fully automated nor scalable.

In [7], Bhatia et al. proposed a Domain–Specific Hidden Web Crawler (DSHWC) that automated the process of downloading of search interfaces, finding the semantic mappings, merging them and filling the Unified Search Interface (USI) produced thereof has been designed that finally submits the form to obtain the response pages from Hidden Web. The DSHWC automated the process of searching, viewing, filling in and submitting the search forms followed by the analysis of the response pages.

Since the Hidden Web has broad and relevant coverage of dynamic [16] and high quality contents, there is a need to refresh/update the local collection of Hidden web documents incrementally [4,11,13],so as to maintain document collection or repository updated with the latest information.

However, a critical look at the available literature [4,6,11,12,14,16] indicates that although the Hidden Web crawling enables the crawler to download Hidden Web but none of the work has been done for the purpose of maintaining the document collection of Hidden Web pages updated with the latest information. So, there is a need of some mechanism to keep the repository fresh. For the purpose, it is required to verify whether a web page has been changed or not. Therefore, a mechanism needs to be introduced for adjusting the time period between two successive revisits of the crawler based on probability of updation [5,15]of the web page. The architecture of an Incremental Hidden Web Crawler (shown Fig. 2) has been proposed that introduces a technique to continuously update/refresh the Hidden Web repository. It uses a mechanism for adjusting the time period between two successive revisits of the crawler based on probability of the web page [5,11,15].

III. PROPOSED WORK

The architecture of an Incremental Hidden Web Crawler (shown Fig. 2) has been proposed that introduces a technique to continuously update/refresh the Hidden Web repository. It uses a mechanism for adjusting the time period between two successive revisits of the crawler based on probability of the web page [5,11,15].

The proposed architecture consists of the following functional components:

1. Domain-Specific Hidden Web Crawler (DSHWC)
2. URL Extractor
3. Revisit Frequency Calculator
4. Update Module
5. Dispatcher

The description of each functional component with required data structures is given below.

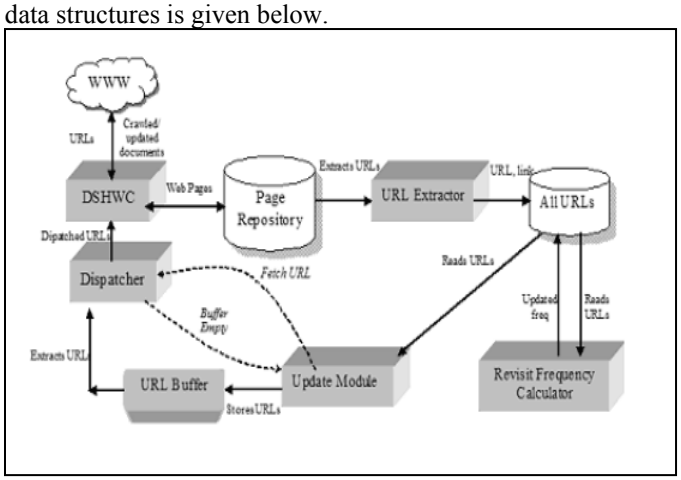

Figure 2. Architecture of an Incremental Hidden Web Crawler

A. Domain Specific Hidden Web Crawler(DSHWC)

DSHWC [7] is a fully automated crawler that downloads search interfaces, finds the semantic mappings, merges them and fills the Unified Search Interface (USI) produced thereof. Finally, the DSHWC submits the form to obtain the response pages from Hidden Web [2,6,10].

After obtaining response pages, the DSHWC stores the downloaded pages into Page repository that maintains the documents crawled/updated by the DSHWC along with their URLs.

B. URL Extractor

It extracts the URLs along with their link information from the above repository and stores them in the AllURLs. The details of link information stored in AllURLs have been discussed in next section.

C. AllURLs

It records all URLs that the crawler has discovered, along with their link information as contained in the Page Repository. The AllURLs contains the URLs and their corresponding link information (see Fig. 3) as follows:

λ_{low}, λ_{mid}, λ_{upper} : boundary conditions for change frequency of pages.

λ_{prev} : change frequency of page at the previous visit

$\lambda_{current}$: current change frequency of page

f_n : current revisit frequency

Δf : change in the frequency of the page

f_{n+1}: adjusted crawler revisit frequency

$D_{last\ crawl}$: date of their last crawl

$T_{last\ crawl}$: time of their last crawl

D_{curr} : current date

T_{curr} : current time

The above mentioned information is further used by the Revisit Frequency Calculator to compute the revisit frequency for DSHWC.

URL	λ_{low}	λ_{mid}	λ_{upper}	λ_{prev}	λ_{curr}	f_n	Δf	f_{n+1}	$T_{last\ crawl}$	$D_{last\ crawl}$	T_{curr}	D_{curr}

Figure 3. URLs and their link information

D. Revisit Frequency Calculator

In order to maintain the freshness of Page Repository, the proposed crawler must download the fresh pages, therefore, Revisit Frequency Calculator [11] of Incremental Hidden Web Crawler finds the appropriate revisit frequency of the crawling so that crawler can update its Page Repository with fresh documents.

In order to compute revisit frequency of the crawling [11], it is necessary to find at which rate crawler needs to visit each page. The rate of revisit of a page should increase for the pages that change more often. However, this may not be the conclusion as frequent revisits of a page, mayn't always provide the updated information. So, there is a need to modify the revising rate of a page. It has been observed from Fig. 4 that the revisiting frequency of the crawler is proportional to the change frequency [5,15] of the page up to a certain threshold value (λ_{middle}), after the threshold it remains constant up to the next threshold (λ_{upper}) and then decrease, with increase in the change frequency of the page after the second threshold (λ_{upper}).

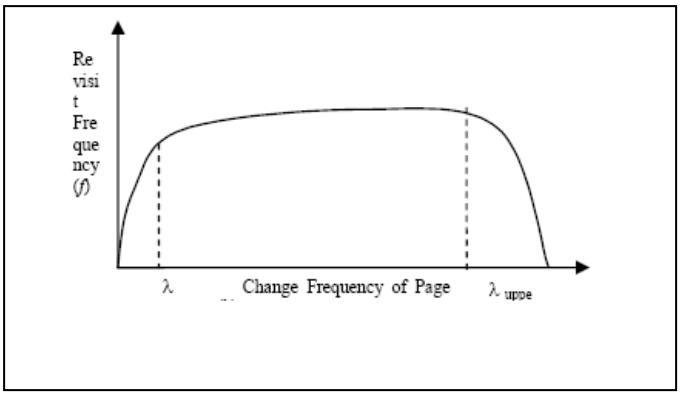

Figure 4. Change frequency of page vs. Revisit frequency

This functional component reads AllURLs and for each URL_i, it computes the Revisit frequency of the crawler by using the following equation:

$$f_{n+1} = f_n + \Delta f \quad (1)$$

where

$$\Delta f = [\{f_n \times (\lambda_{current}/\lambda_{previous} - 1) \times u(\lambda_{current} - \lambda_{lower}) \times u(\lambda_{middle} - \lambda_{current}) \times u(\lambda_{upper} - \lambda_{current}) + \{f_n \times (1 - \lambda_{current}/\lambda_{upper}) \times u(\lambda_{current} - \lambda_{upper}) \times u(1 - \lambda_{current}\}] \quad (2)$$

and u(x) is a unit step function.

On the calculation of the adjusted crawler revisit frequency, AllURLs needs to be updated. Also, Δf, f_n, $\lambda_{previous}$, $\lambda_{current}$ needs to be updated. The algorithm for the Revisit frequency calculator is as follows:

```
RevisitFrequencyCalculator( )
{
For each URL_i in AllURLs
{
compute Δf;
compute (f_{n+1} = f_n + Δf);
f_n = f_{n+1} ;
λ_prev = λ_curr ;
update AllURLs with (Δf, f_n ,f_{n+1}, λ_prev, λ_curr);
}
}
```

For each URL in AllURLs, the revisit frequency f_{n+1} has been computed by using the above mentioned method. This revisit frequency is further overwritten by the calculator in AllURLs to previous frequency i.e. f_n.

E. The Update Module

Update module one by one fetches the URLs and their updated link information (with updated revisit frequencies) from the AllURLs. After fetching a URL from AllURLs, it finds the time, denoted as τ (computed as inverse of revisit frequency), after which crawler should revisit the same web page and compare it with a threshold time T. If τ of a given URL is greater than T, then the web page needs to be recrawled and this URL is further stored in URL Buffer thereof, otherwise this URL is discarded and will not be stored in URL Buffer but it will exist in AllURLs. In this way, the URLs of all the web pages that need to be recrawled will be

placed in the URL Buffer. When this buffer will become full, the Update Module will send a signal called Fetch URL signal to Dispatcher. After getting the Fetch URL signal from Update Module, the Dispatcher will start fetching the URLs one by one and forward it to the crawler to download the web page again, so that the freshness of Page repository can increase. When URL Buffer will become empty, the Dispatcher will in turn send Buffer Empty signal to Update Module. Buffer Empty signal signifies that currently the buffer is empty and after receiving this signal, Update Module will process and add more URLs in the URL Buffer. The algorithm given below describes how the Update Module makes the decision for recrawling the selected URLs, which in turn maintains the repository fresh.

```
Update ( )
{
wait (Buffer Empty);
fetch URLs and their link info from AllURLs;
for each fetched URL do
{
calculates τ ;
if (τ >= T) then
add URL in URL Buffer;
else
continue;
if full(URL Buffer) then
signal (fetch URL);
}
}
```

The revisit frequency for URL *www.sample.com* has been computed by assuming the following values.

$\lambda_{lower} = 0.1$, $\lambda_{middle} = 0.6$ *and* $\lambda_{upper} = 0.8$,

$D_{last\ crawl}$ = *13-01-2010,* $T_{last\ crawl}$ = *14:00.*

Now consider the data given below:

Initial frequency of revisit f_l = 10 times/unit time.

Current change frequency of page $\lambda_{current}$ = 0.15 and the change frequency at the previous visit $\lambda_{previous}$ = 0.1.

Computing Δf as given below:

$\Delta f = [\{10\times(0.15 / 0.1 -1)\times u\ (0.15 – 0.1)\times u\ (0.6- 0.15)\times u\ (0.8-0.15)\} + \{10\times(1-0.15 / 0.1)\times u\ (0.15 – 0.8)\ \times(1-0.15)\}]$.

$\Delta f = 5$

Now, the new revisit frequency can be computed as follows.

$f_{n+1} = 10+5 = 15$ *times/unit time.*

Let the basic unit of revisit be 24 hours and T (threshold time) =2 hours.

Calculating τ *(time) =2(approx.)*.

It means that the crawler will revisit the corresponding web page every 2 hours (i.e. after a time period of 2 hours).

Now, τ is compared with T. If τ is greater than or equal to T, then the corresponding web page needs to be recrawled otherwise not. Since, for the above example τ is equal to T, so the web page needs to be recrawled and is further added in the URL Buffer.

F. Dispatcher

Dispatcher waits for the *fetch URL* signal and upon receiving this signal, it fetches an URL from the URL Buffer so that DSHWC in turn can download the corresponding web page. However, if dispatcher finds the URL Buffer empty during this operation, then it sends Buffer Empty signal to the Update Module so that it can add more URLs in the URL Buffer. The algorithm for Dispatcher is given below.

```
Dispatcher ( )
{
wait (fetch URL);
while (not (empty URL Buffer)
{
fetch URL from URL Buffer;
forward URL to DSHWC to download web page;
}signal (Buffer Empty);}
```

IV. IMPLEMENTATION AND PERFORMANCE EVALUATION

With the increase in the availability of hidden web pages, the major problem faced by the current search engines [3] is difficulty in fresh or updated information retrieval. It is problematic to identify the desired information from amongst the large set of web pages given by the search engine. With further increase in the size of the Hidden Web [2,6,10] contents, the problem grows exponentially. The number of web pages which have gone under updation increases [1, 12, 16], with the increase in web size. As discussed above, the proposed incremental hidden web crawler downloads the fresh hidden web contents incrementally to make the search engine repository updated. The proposed incremental Hidden Web crawler has been implemented using .Net technology on Windows platform and the snap shot for the incremental hidden web crawler has been shown in Fig.5. Several experiments were conducted over the *Books* and *Airline* domains and the initial results were very promising.

The proposed incremental hidden web crawler updates the hidden web pages that are already crawled by the hidden web crawler and makes the repository fresh. Therefore, to evaluate the proposed work, two performance metrics i.e. *freshness of database* and *age of database* are taken in to the consideration and several experiments were conducted to evaluate freshness and age of database.

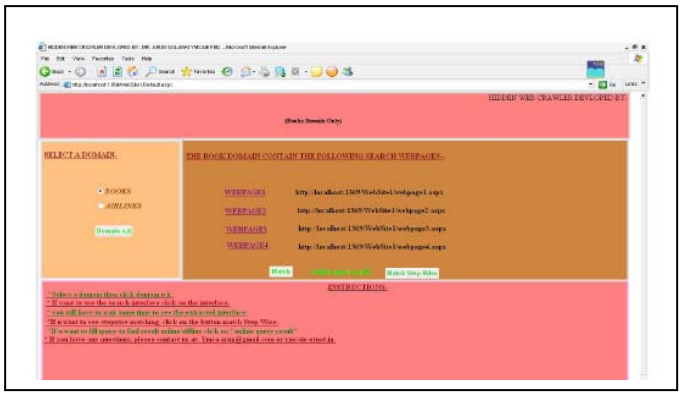

Figure 5. Snapshot for Domain-specific Hidden Web Crawler

Freshness of database D at time t is computed as

$$F(D, t) = 1/N \sum F(e_i, t) , \quad (3)$$

where

$$F(e_i, t) = \begin{cases} 1, \text{ if page } e_i \text{ is up-to-date at time} \\ 0, \text{ otherwise} \end{cases}$$

and N is total number of web pages in D.

Similarly, Age of database D at time t is computed as

$$A(D, t) = 1/N \sum A(e_i, t) , \quad (4)$$

Where

$$A(e_i, t) = \begin{cases} 0, \text{ if page } e_i \text{ is up-to-date at time t} \\ t \text{ ,modification time of } e_i, \text{ otherwise} \end{cases}$$

and N is total number of web pages in D.

As discussed above, that proposed incremental hidden web crawler updates the repository created by the hidden web crawler, therefore, freshness and age of the updated database has been verified and analyzed. The observations show that as more fresh pages are crawled by the proposed crawler, the freshness of the database increases and the age of the database decreases with increase in time i.e. the database become fresher as the updated hidden web pages are recrawled by the incremental hidden web crawler.

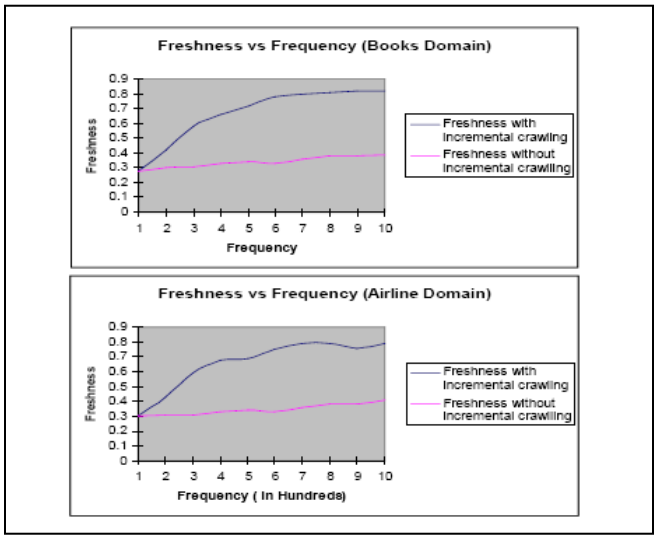

Figure 6 (a) & (b). Freshness vs. Frequency for Book and Airlines Domain

Therefore, it may be concluded that the age of the database decreases as freshness of the database has been increased. The observation has been shown in form of graphs for both the domains in Fig. 6 (a) and (b).

V. CONCLUSION

In this paper, design of an incremental Hidden Web Crawler has been discussed that not only crawls the Hidden Web but also maintains the repository updated with the new/ updated pages. It is based on adjusting the time period between the two successive revisits of the crawler based on probability of updation of a web page. The experiments conducted over real web sites indicate that crawler always keeps the repository fresh with high freshness rate.

As the future work, the architecture of a search engine based on Incremental Hidden Web crawler can be designed. Moreover, some indexing technique may be applied to index the web pages stored in the repository.

REFERENCES

1. Arvind Arasu, Junghoo Cho, Hector Garcia-Molina, Andreas Paepcke, and Sriram Raghavan, "Searching the Web", ACM Transactions on Internet Technology (TOIT), 1(1):2–43, August 2001.
2. Michael K. Bergman, "The deep web: Surfacing hidden value", Journal of Electronic Publishing, 7(1), 2001.
3. Sergei Brin and Lawrence Page, "The anatomy of a large-scale hypertextual Web search engine", Computer Networks and ISDN Systems, 30(1–7):107–117, April 199
4. J. Cho and H. Garcia-Molina. "The Evolution of the Web and Implications for an Incremental Crawler." In Proceedings of the Twenty-Sixth VLDB Conference, pp. 200– 209, Cairo, Egypt, 2000.
5. J. Cho and H. Garcia-Molina. "Estimating Frequency of Change." Technical report, DB Group, Stanford University, Nov 2001.
6. Czajkowski, K., Fitzgerald, S., Foster, I., Kesselman, C.: Grid Information Services for Distributed Resource Sharing. In: 10th IEEE International Symposium on High Performance Distributed Computing, pp. 181--184. IEEE Press, New York (2001) [6] S. Raghavan and H. Garcia-Molina, "Crawling the Hidden Web", In Proc. of VLDB, pages 129–138, 2001.

7. Komal Kumar Bhatia, A.K.Sharma, "A Framework for an Extensible Domain-specific Hidden Web Crawler (DSHWC)", communicated to IEEE TKDE Journal Dec 2008.
8. Komal Kumar Bhatia, A.K.Sharma, "A Framework for Domain-Specific Interface Mapper (DSIM)", International Journal of Computer Science and Network Security (IJCSNS 2008).
9. Komal Kumar Bhatia, A.K.Sharma, "Merging Query Interfaces in Domain-specific Hidden Web Databases", accepted in International Journal of Computer Science, 2008.
10. A.K. Sharma, Komal Kumar Bhatia, "Crawling the hidden web resources", Proc. of NCIT- 2007, Delhi.
11. Ashutosh Dixit and A.K Sharma, "Self Adjusting Refresh Time Based Architecture For Incremental Web Crawler", International Journal of Computer Science and Network Security (IJCSNS), Vol 8, No12, Dec 2008.
12. Mike Burner, "Crawling towards Eternity: Building an archive of the World Wide Web", Web Techniques Magazine, 2(5), May 1997.
13. Junghoo Cho and Hector Garcia-Molina. 2000, "The evolution of the web and implications for an incremental crawler". In Proceedings of the 26th International Conference on Very Large Databases
14. A. K. Sharma, J. P. Gupta, D. P. Agarwal, " A novel approach towards management of Volatile Information" Journal of CSI, Vol. 33 No. 1, pp 18-27, Sept' 2003.
15. Junghoo Cho and Hector Garcia-Molina. Estimating frequency of change, 2000.Submitted to VLDB 2000, Research track.
16. Brian E. Brewington and George Cybenko. "How dynamic is the web.", In Proceedings of the Ninth International World-Wide Web Conference, Amsterdam, Netherlands, May 2000.

Rosy Madaan received B.E. degree in Computer Science & Engineering with Hons. from Maharshi Dayanand University in 2005 and is persuing M.Tech. in Computer. Presently, she is working as Senior Lecturer in Computer Engineering department in B.S.A. Institute of Technology & Management, Faridabad. Her areas of interests are Search Engines, Crawlers and Hidden Web.

Ashutosh Dixit received the B.E, M.Tech. degrees in Computer Science Engineering with Hons. from Maharshi Dayanand University in 2001 and 2004 respectively. Presently, he is working as Senior Lecturer in Computer Engineering department in YMCA University of Science & Technology, Faridabad. He is also persuing Ph.D in Computer Engineering and his areas of interests are Search Engines and Crawlers.

Prof. A. K. Sharma received his M.Tech. (Computer Science & Technology) with Hons. from University of Roorkee in the year 1989 and Ph.D (Fuzzy Expert Systems) from JMI, New Delhi in the year 2000. From July 1992 to April 2002, he served as Assistant Professor and became Professor in Computer Engg. at YMCA University of Science & Technology, Faridabad in April 2002. He obtained his second Ph.D. in IT from IIIT & M, Gwalior in the year 2004. His research interests include Fuzzy Systems, Object Oriented Programming, Knowledge Representation and Internet Technologies.

Dr. Komal Kumar Bhatia received the B.E, M.Tech. and Ph.D. degrees in Computer Science Engineering with Hons. from Maharshi Dayanand University in 2001, 2004 and 2009, respectively. Presently, he is working as Assistant Professor in Computer Engineering department in YMCA University of Science & Technology, Faridabad. He is also guiding Ph.Ds in Computer Engineering and his areas of interests are Search Engines, Crawlers and Hidden Web.