Hidden Data Extraction Using URL Templates Processing

Babita Ahuja, Anuradha and Dimple Juneja

Abstract A lot of work has been carried out in the deep web. Deep web is like a golden apple in the eyes of the researchers. Most of the deep web search engines extract the data from the deep web, store them in the database, and index them. So, such kind of techniques have the disadvantage of less freshness, large repository requirement and need of frequent updating of the deep web database to give accurate and correct results. In order to overcome these drawbacks, we propose a new technique "Hidden Data Extraction using URL Template processing" where the fresh results from the website server database are fetched dynamically and are served to the users.

Keywords Search engines · Surface web · Hidden web · Query interfaces

1 Introduction

Most of the websites store their data in the databases. These databases contain high quality and a large amount of data [1]. The data in the databases are not accessible directly by the traditional search engines. The user has to fill the query interfaces in order to retrieve the data in the databases. These query interfaces act as a blockage

11th International Conference on Wirtschaftsinformatik, 27th February–01st March 2013, Leipzig, Germany.

B. Ahuja (✉)
MRU, Faridabad, India
e-mail: babitaspark@mru.edu.in

Anuradha
YMCAUST, Faridabad, India
e-mail: anuangra@yahoo.com

D. Juneja
DIMT Kurukshetra, Faridabad, India
e-mail: dimplejunejagupta@gmail.com

V.B. Aggarwal et al. (eds.), *Big Data Analytics*, Advances in Intelligent Systems and Computing 654, https://doi.org/10.1007/978-981-10-6620-7_13

in accessing the deep web data [2]. A major part of the hidden web data is behind the query interfaces. Many researchers have worked on it. However, major of the techniques suffer from the problem of freshness, bulk database requirement and frequently crawling the web for hidden data. When users issue the query in these techniques, the deep web data residing in the local database repository of search engines are displayed to the user. The deep web local database loses its freshness in a few minutes and even in few seconds.

1.1 Traditional Method to Uncover Deep Web

When user wants to access the deep web, he has to fill multiple query interfaces as shown in Fig. 1. Steps taken by user to access deep web in traditional search engines are:

Step 1. User fills the single search text field query interface of search engine.
Step 2. The result web pages from deep web and surface web are displayed to user. The deep web data contains the stale web pages stored in search engine local repository and the query interfaces which act as an entry point for accessing deep web.
Step 3. In order to access the fresh data, user opens the query interfaces. User fills all the text fields of the query interface and submits the page. The result is displayed to user.

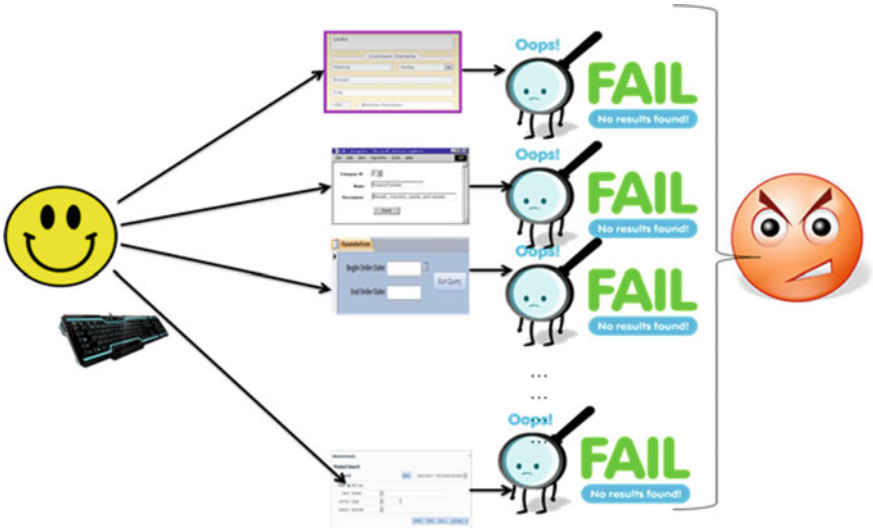

Fig. 1 Traditional way to see deep web

Step 4. User recursively repeats the same step 3 until desired results are retrieved. Filling the same values in thousands of query form is tedious and monotonous. Therefore, user finally quits and is unsatisfied with the results.

1.2 Steps of Proposed Technique to Uncover Deep Web

In the proposed technique, the user is not required to fill all the query interfaces. User will fill a single search text field for his query as shown in Fig. 2. The steps taken by user in proposed technique to access deep web are:

Step 1. User fills the single search text field query interface of search engine.
Step 2. The user query is processed.
Step 3. User keywords are placed in the URL templates and dynamic URL is generated.
Step 4. For post methods of form, the user keywords are embedded in the source code of the page and are submitted.
Step 5. The results are fetched on the fly from the website servers. These fresh results are displayed to the user.

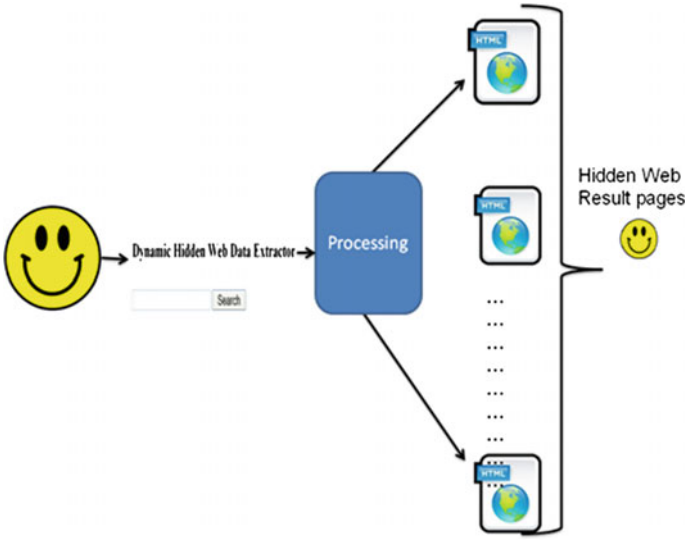

Fig. 2 Steps taken by user in proposed technique to fetch deep web pages

2 Related Work

Manuel Alvarez developed a hidden web crawler called DeepBot [4] to access Deep web. DeepBot has a "mini-web browser" to handle client-side scripts and session. Maintaining mechanism. Sriram has created a Hidden web crawler called HiWE [5]. HiWE stands for Hidden Web Exposer. HiWE is a task-specific hidden web crawler. HiWE extracts the data hidden behind the web query interfaces. Dr. Komal Kumar Bhatia proposed an incremental web crawler [6]. The incremental web crawler continuously refreshes the Web Repository. Ntaulas developed a search engine called HiddenSeek [7]. HiddenSeek works on the single-attribute database. It also detects the spam websites and ranks the pages also. Dr. Anuradha has created a Hidden Web Search Engine [8]. The hidden web search engine auto-fills the web query interfaces, extracts the result records, store them in a repository for later searching. Mining data records (MDR) [9] proposed by Chen. MDR searches for relevant data by looking for the form and the table tag in the web page. The "Layout-Based Data Region Finding" is a wrapper technique proposed by Chen [10]. The advantages and disadvantages of different techniques are given below in Table 1.

Table 1 Comparison of different hidden web extraction tools

	Type of tool	Method	Advantages	Disadvantages
1	Hidden web crawlers	DeepBot	It deals with both client-side scripting code and server side deep web data	Domain definitions are required, need mass storage
2	Hidden web crawlers	HIWE	It extracts the data from hidden databases	Need human assistance to fill the forms, need mass storage
3	Hidden web crawlers	Incremental web Crawler	It calculates the time to revisit and hence no unnecessary crawl is required	The page repository needs to be updated very frequently for few pages
4	Hidden web search engine	HiddenSeek	Uses different query selection policies, handles spam websites	Work on single-attribute databases, mass storage required
5	Hidden web search engine	Hidden web search engine	It works on multi-attribute interfaces, data is compiled and service is given to user	Millions of queries and Mass storage is required
6	Wrapper technique	Mining data records	It can mine the data records in a table automatically	Some advertisement and irrelevant records are also fetched
7	Wrapper technique	Layout-based data region finding	Constructs tree for getting data	The tags like , are not considered

3 Proposed Work

We propose a new technique "Hidden Data Extraction behind Query Interfaces" shown in Fig. 3. In the proposed system on the system side, the query interfaces are extracted and categorized on the basis of domain and the form submission method. The URL of these query interfaces are analyzed thoroughly. After analyzing, the query interfaces the URL templates are created and are stored in the URL templates repository.

For query interfaces having the Post method of form submission, the source code of the query interfaces is updated in order to fetch the fresh results from web servers. On the end user side when user will issue the query. The user query will be processed and keywords will be extracted. These keywords are placed in the URL templates of query forms having GET method of submission. In the case of POST method of submission, the keywords are placed in the source code of the query forms. After that, the results behind the query interfaces are pulled out. These results pages are then displaced to the user. Modules of the proposed.

Algorithm:

1. Query Interface Extraction.
2. URL Template Extraction of Query Interfaces having "GET" method of form submission.

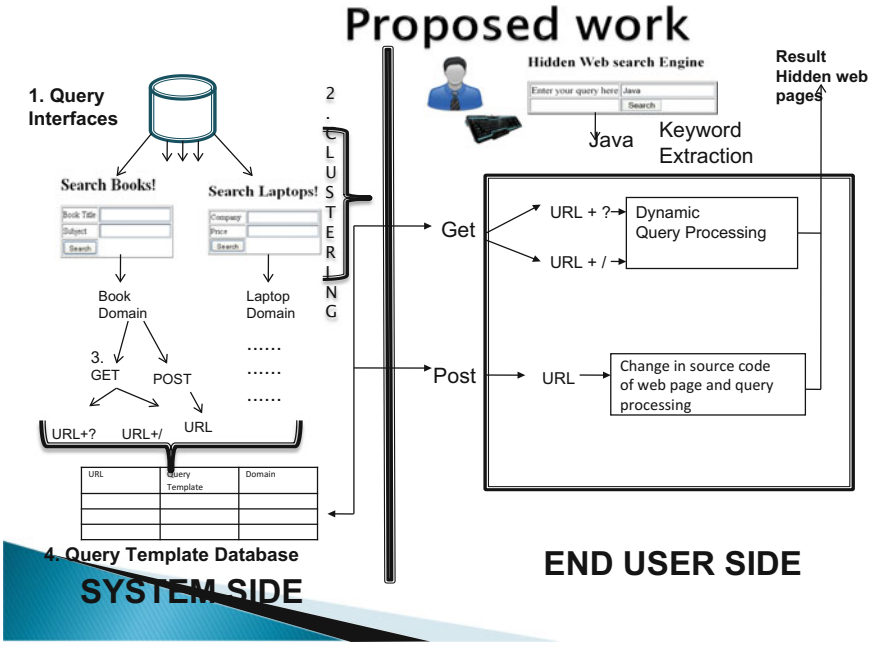

Fig. 3 Proposed architecture of hidden data extraction behind query interfaces

3. For POST method of submission extract the result page URL.
   3.1. Extract the query form of the page.
   3.2. Change the source code of the page.
4. Process the user query.
5. Create the dynamic URL of GET method Query Interfaces and fetch the results.
6. Set the values of user query in the source code of the post query interface and auto submit the page. Fetch the results.
7. Show 10 pages having maximum page rank. For these web pages and the other web pages calculate the fresh page rank. The pages are then ordered on the basis of the page rank and displayed to the users.

3.1 Query Interface Extraction

The data behind the query interfaces forms a lion's share of the data on the hidden web. The query interfaces act as a very significant channel to access the databases residing on the server machines databases. These query interfaces are created on computer machines but the irony is that the computer machines are not able to understand them. So fetching the query interfaces is a major task. The query interfaces can have two form processing techniques GET and POST. Most of the techniques developed consider only the GET method of form submission. The proposed technique works on both the GET and the POST. In the query interface extraction module we will fetch the query interfaces of few sample domains. In order to fetch the query interfaces, we will issue the query using the Google API. Here we will consider two domains book and the car domains. When a query is fired using google API ex: "book" then results of book domain are displayed. These result pages contain static web pages as well as pages containing query interfaces. The pages containing the query interfaces are filtered and are stored. The stored query interfaces are further categorized on the basis of GET and POST.

Algorithm:

```
1. Initializing an array Domains which can be of n size i.e. Domains d[n]
2. for each  di  send request to google using API
3.    for each result page pj returned from google API repeat 4 to 8
4.      if (pj .contains( method="GET") )
5.        Add to Database db1: PageURL pj, Domain di, GET Method
6.      else if (pj .contains( method="POST") )
7.         Add to Database db1: PageURL pj, Domain di, POST Method
8.      else discard pj
```

3.2 URL Template Creation

The URL filtered above contains the query interfaces. The query interfaces having GET method of form submission will be handled in this module. When these query interfaces are submitted to servers the result pages are shown. In GET as the input field data changes so does the result page's URL changes. The URL template of these result pages will be fetched and will be stored in the database. In order to accomplish this task the form tag of the query interfaces will be extracted. A temporary web page will be created which will contain the code of form tag extracted in the previous step. The websites provide the relative path of the result pages. Therefore, these relative paths will be converted into the absolute path. The initial input field will be filled with any value. The temporary web page will be created in such a way that it will automatically be redirected to the result page. The URL of the result page will be extracted. The URL will be analyzed and a generic template of the URL will be created and stored in the URL template database.

Algorithm:
URL Template Creation (DB_URL pages [ ])
// extracting the URL pattern of GET method web pages
1. for each $Page^i$ in DB1 where method= "GET" repeat 2 to 8
2. Extract the form tag in the web page
3. Get the action attribute of <form> tag
4. Change the relative URL to the absolute URL
5. Fill any value in the first input field of the form only.
6. write to updated form in a file "temp.html"
7. resultUrl=autosubmit(temp.html)
8. Analyze resultUrl and create Url Template
9. Add to DBTemplete : Url Template, domain, method

3.3 Template Extraction

The query interfaces extracted in the first step contains web pages having both GET and POST methods of form submission. In GET as the input field data changes so does the result page's URL changes. In GET URL template is stored. However, in the POST method, the result page's URL does not change on the change of the input field data. For every query interface having POST method of form submission, a separate html page will be created. This html page contains only the form tag data of the original query interface. The page will be created in such a way that the input field values will be filled atomically by the value that is provided by the user while issuing the query in a single search textbox. The values will be filled

automatically and the submission too will be done automatically. The fresh results from the actual website's servers will be presented to the user.

Algorithm:
1. for each Page^{i} in DB where method= "POST" repeat 2 to 5
2. Extract the form tag in the page
3. The values of text field in the form are replaced by a dynamic value.
4. The updated form tag is written to a HTML file "formpage^{i} .html".
5. Add to DBTemplate: formpage^{i} .html, domain, method.

3.4 User Query Processing

The user will issue the query in a single search text field. The user query will be processed. The tokens or keywords will be extracted from the user query. The punctuation symbols, etc., will be removed from the user query. The lemmatization or stemming of user query will be done to formulate the final list of user query keywords. The keywords will be processed to identify the domain of the user query.

Algorithm:
1. Keywords [] =split ("query"," ")
2. Remove stop words from keywords
3. Stemming of keywords
4. Identify domain of user query

3.5 Query Interfaces Having GET Method of Submission

All the URL templates of user query domain will be picked from URL template database. The user keywords will be placed in the URL templates and the form input data fields. The URL's will be created and will be displayed to the user. When user will click on the URL then the result pages from the actual website will be fetched and will be displayed to the user.

Algorithm:
1. URLs extracted from URL template database
2. for every URL^i repeat 3 to 6
3. if(URL^i.method="GET")
4. { // Generate dynamic query string
5. add keywords in URL template
6. Result_set1= dynamic URL's created }

3.6 Query Interfaces Having POST Method of Submission

The web pages having POST method of submission will be handled here. The web pages created in module 3 template extraction, will be updated and the values from the user query keywords will be placed in the text field of the web page.

Algorithm:
1. if (URL^i.method="POST")
2. { // open the files created
3. open $formpage^i$.html
4. $formpage^i$.setValues(form_fields, keywords[])
5. autosubmit($formpage^i$.html)
6. Result_set2= fetch result pages
7. Display pages (result_set1, result_set2)}

4 Experimental Results

The admin on the system side will extract the URLs of the query interfaces of particular domain. The generic query templates of these URL have been created and are stored in the database as shown in Fig. 4. After the query interface extraction, the generic templates of the result pages will be created as shown in Fig. 5. These generic templates are stored in the local repository of the system. When user query in the single search text field the query is processed and the results are displayed to the user as shown in Figs. 6 and 7.

Fig. 4 Query interface extraction

5 Comparison with Other Search Engines

The parameters on which the comparison is done are relevancy and optimized results. Relevancy means how much relevant result pages according to the user query are fetched from deep web servers. The implementation results show that deep web pages fetched by the proposed work are highly relevant as compared to other search engines as shown in Figs. 8, 9 and 10.

Another factor taken into consideration for comparison is the optimum result. The results shown by another search engine in terms of price of the book or any other item are not optimum as shown in Figs. 11, 12, 13 and 14.

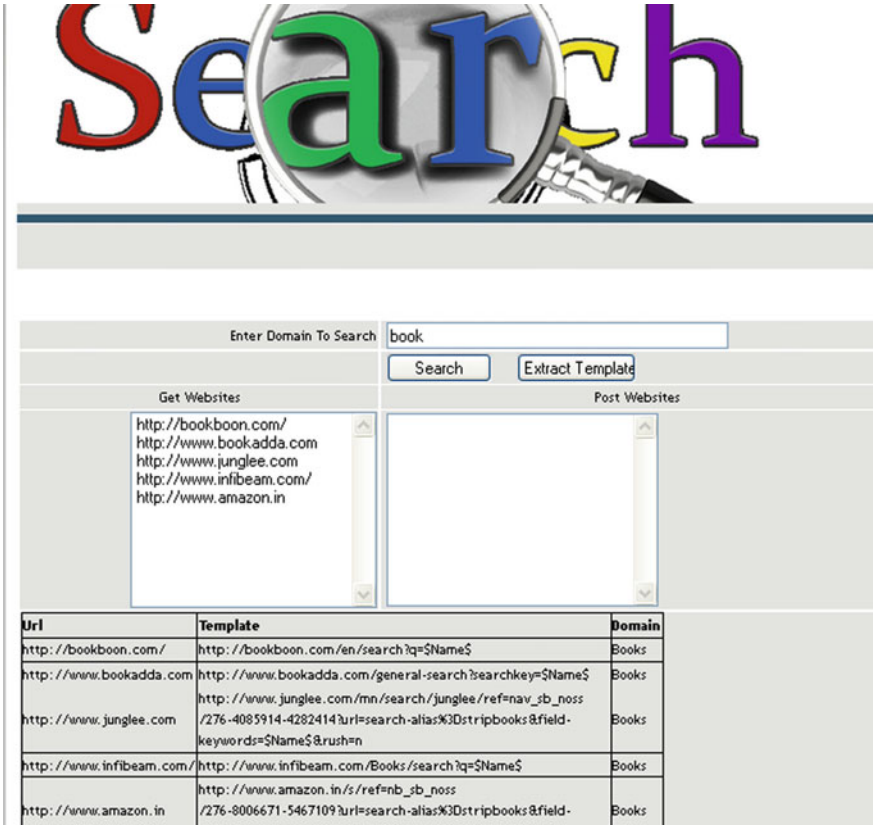

Fig. 5 Template extraction

Fig. 6 Result page displayed to user

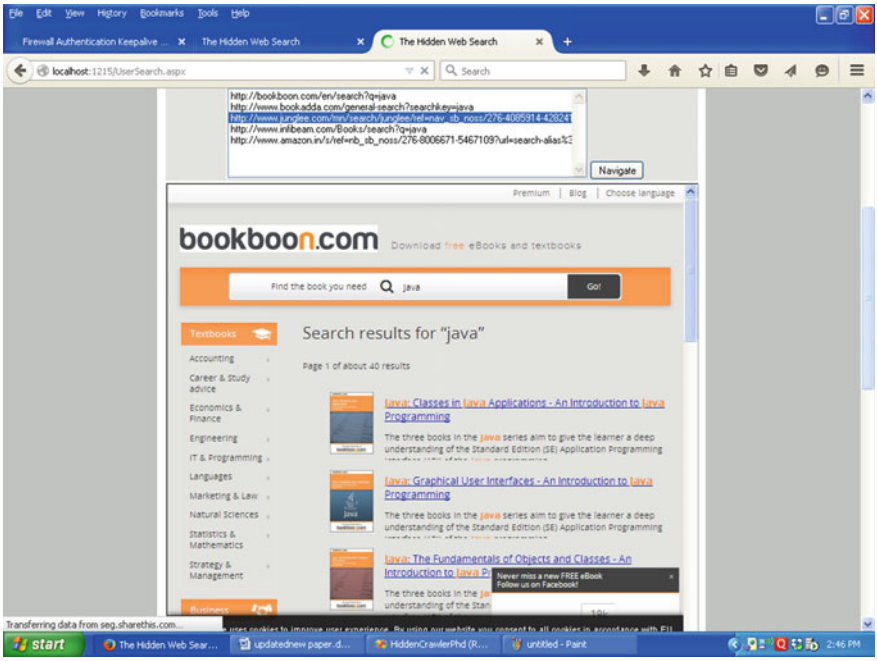

Fig. 7 Deep web data from the website presented to the user

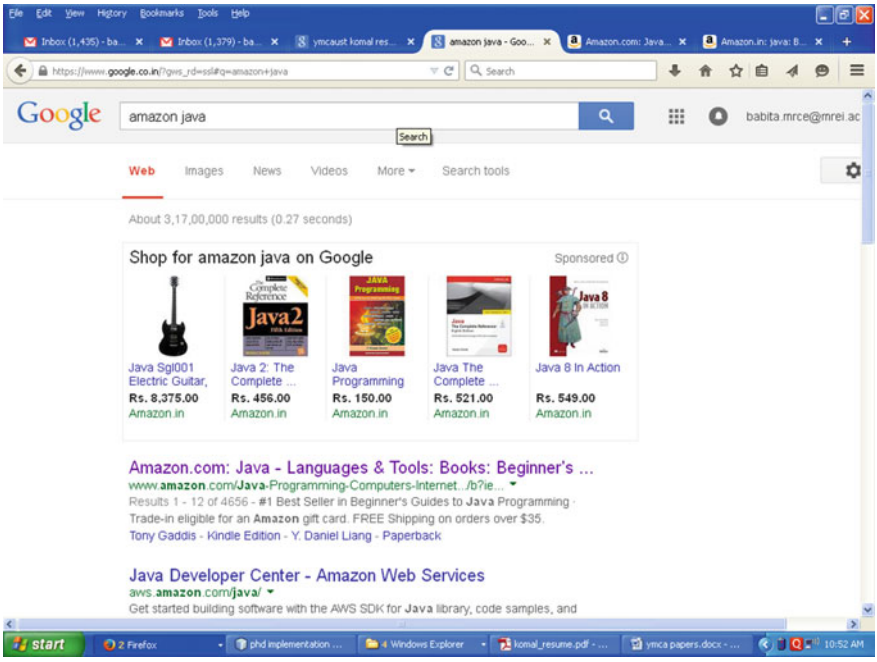

Fig. 8 Query fired on other search engine

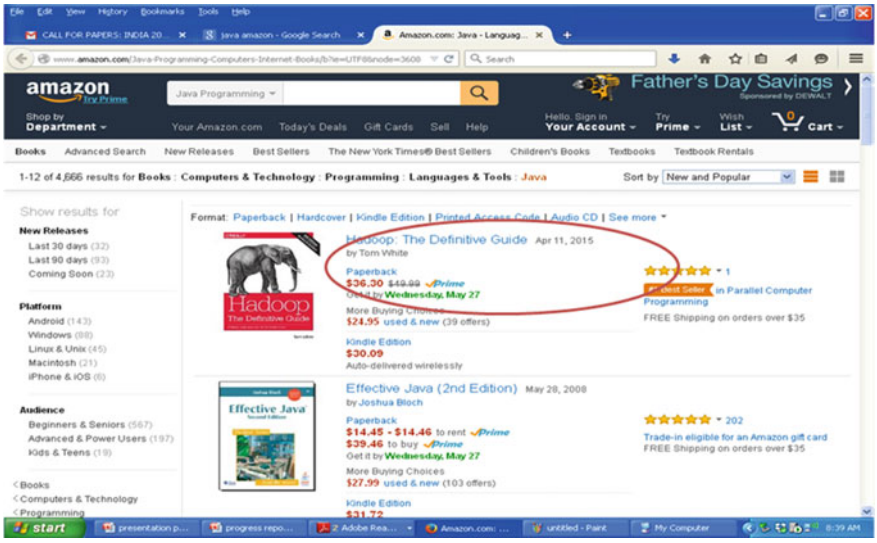

Fig. 9 Irrelevant result shown by other search engine

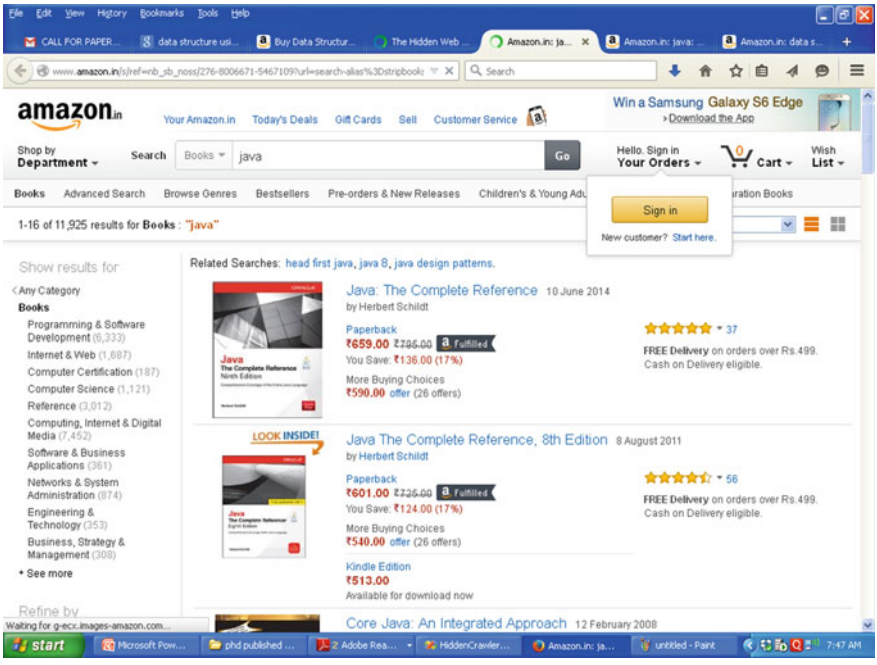

Fig. 10 Relevant data shown when query issued by proposed work

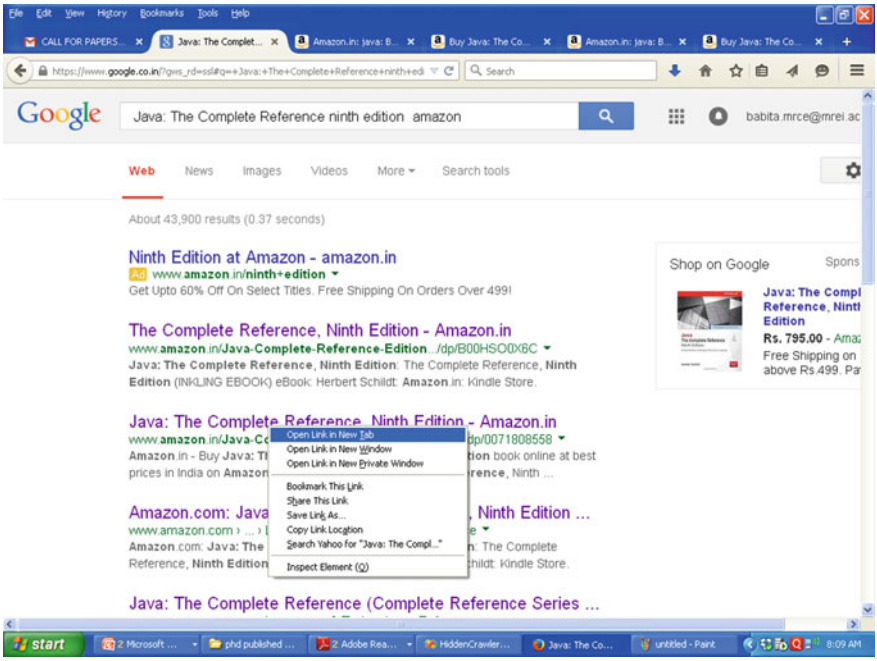

Fig. 11 Query fired on other search engine

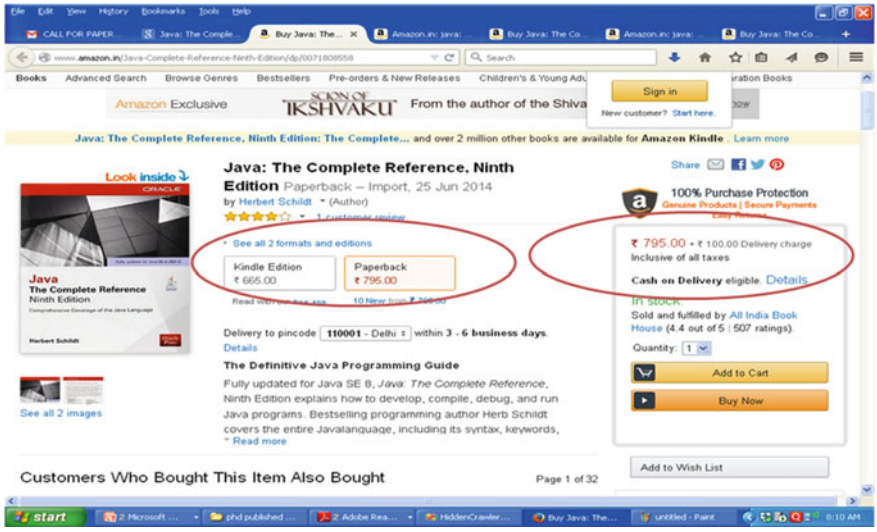

Fig. 12 High price book shown by other search engine

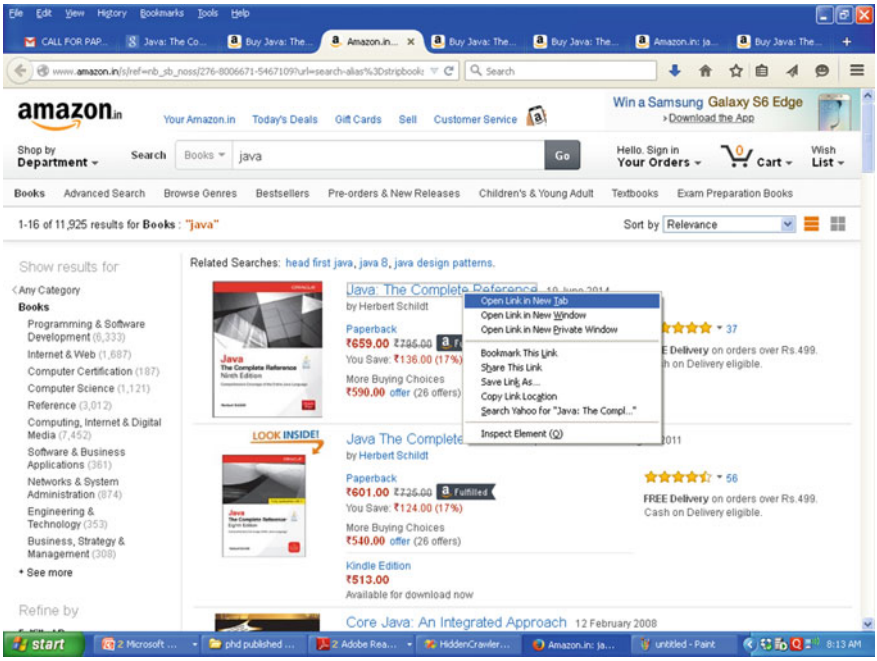

Fig. 13 The result shown by proposed work

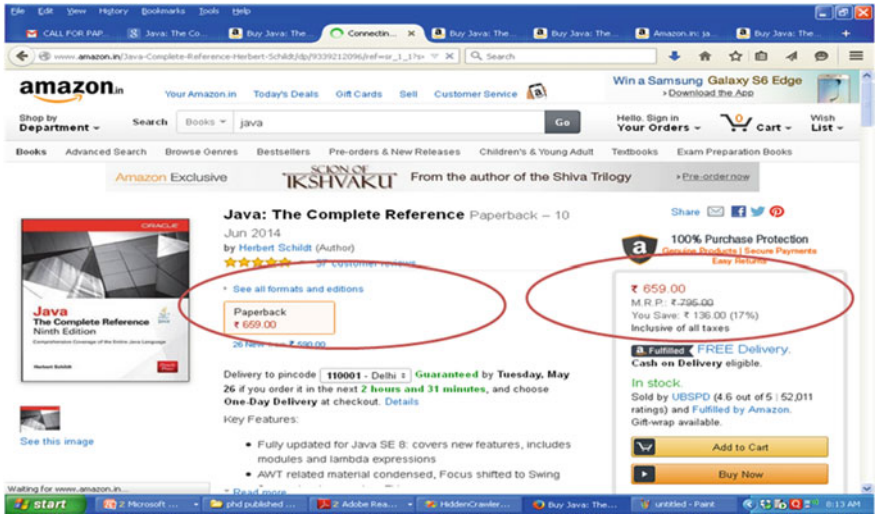

Fig. 14 The low price book shown by proposed work

References

1. BrightPlanet.com. The deep web: surfacing hidden value. Accessible at http://brightplanet.com, July 2000
2. Sherman, C., Price, G.: Hidden Web. Uncovering Information Sources Search Engines Can't See. CyberAge Book (2001)
3. Bergman, M.K.: White paper. The deep web: surfacing hidden value. J. Electron. Publ. **7**(1) (2001)
4. Álvarez, M., Raposo, J., Cacheda, F., Pan, A.: A task-specific approach for crawling the deep web. Eng. Lett. **13**(2), EL_13_2_19 (Advance online publication: 4 Aug 2006)
5. Raghavan, S., Garcia-Molina, H.: Crawling the hidden web. VLDB, (2001)
6. Madaan, R., Dixit, A., Sharma, A.K., Bhatia, K.K.: A framework for incremental hidden web crawler. Int. J. Comput. Sci. Eng. **02**(03), 753–758 (2010)
7. Ntoulas A., Zerfos, P., Cho, J.: Downloading hidden web content. Technical Report, UCLA
8. Anuradha, Sharma, A.K.: Design of hidden web search engine. Int. J. Comput. Appl. **30**(9) (2011) (0975-8887)
9. Chen, H.-P., Fang, W., Yang, Z., Zhuo, L., Cui, Z.-M.: Automatic Data Records Extraction from List Page in Deep Web Sources; 978–0-7695-3699-6/09 c, pp. 370–373. IEEE (2009)
10. Liu, B., Grossman, R., Zhai, Y.: Mining data records in web pages. In: KDD'03: Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 601–606, New York