Crawling the Hidden Web Resources: A Review

Rosy Madaan¹, Ashutosh Dixit² and A.K. Sharma²

Abstract

An ever-increasing amount of information on the Web today is available only through search interfaces. The users have to type in a set of keywords in a search form in order to access the pages from certain Web sites. These pages are often referred to as the Hidden Web or the Deep Web. Since there are no static links to the Hidden Web pages, search engines cannot discover and index such pages and thus do not return them in the results. 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. In this paper, the study is made how an effective Hidden Web crawler can be built that can autonomously discover and download pages from the Hidden Web. Since the only “entry point” to a Hidden Web site is a query interface, the main challenge that a Hidden Web crawler has to face is how to automatically generate meaningful queries to issue to the site. Here a theoretical framework to investigate the query generation problem is provided for the Hidden Web and effective policies for generating queries automatically have been studied. The policies proceed iteratively, issuing a different query in every iteration. The effectiveness of these policies is experimentally evaluated on 4 real Hidden Web sites and the results are very promising. For instance, in one experiment, one of the policies downloaded more than 90% of a Hidden Web site (that contains 14 million documents) after issuing fewer than 100 queries.

Key words: Crawler, Invisible web, Hidden web, Surface web, Search interfaces

INTRODUCTION

The terms “invisible web”, “hidden web” and “deep web” all refer to the same thing: a massive storehouse of online data that the search engines don’t capture. The fact that search engines only search a very small portion of the web make the Invisible Web a very tempting resource. There’s a lot more information out there that can be imagined that’s because terabytes of information are buried in databases and other research resources. Searchable and accessible online but often ignored by conventional search engines, these resources exist by the thousands. Known in research circles as the invisible, deep, or hidden Web, this buried

¹. Department of Computer Engineering, Rawal Institute of Engineering and Technology, Faridabad, Haryana.
². Department of Computer Engineering, YMCA University of Science and Technology, Faridabad, Haryana.
content is an estimated 500 times larger than the surface Web, which is estimated at over four billion pages. This mass of information represents a potent research resource, no matter about discipline or interest. Spiders meander throughout the Web, indexing the addresses of pages they discover. When these software programs run into a page from the Invisible Web, they don’t know quite what to do with it. These spiders can record the address, but the information in the page could not told by them. There are a lot of factors, but mainly they boil down to technical barriers and/or deliberate decisions on the part of the site owner(s) to exclude their pages from search engine spiders. For instance, university library sites that require passwords to access their information will not be included in search engine results, as well as script-based pages that are not easily read by search engine spiders.

Deep Web resources may be classified into one or more of the following categories:

1. Dynamic content—dynamic pages which are returned in response to a submitted query or accessed only through a form.
2. Unlinked content—pages which are not linked to by other pages, which may prevent Web crawling programs from accessing the content. This content is referred to as pages without backlinks (or inlinks).
3. Limited access content—sites that require registration or otherwise limit access to their pages (e.g., using the Robots Exclusion Standard), prohibiting search engines from browsing them and creating cached copies.
4. Scripted content—pages that are only accessible through links produced by JavaScript and Flash which require special handling.
5. Non-text content—multimedia (image) files, Usenet archives and documents in non-HTML file formats such as PDF and DOC documents (although some search engines, such as Google, are able to index these types of files).

Search engines crawlers and indexing programs have overcome many of the technical barriers that made it impossible for them to find and provide invisible web pages. These types of pages used to be invisible but can now be found in most search engine results:

- Pages in non-HTML formats (PDF, Word, Excel, Corell suite, etc.) are “translated” into HTML now in most search engines and can “seen” in search results.
- Script-based pages, whose links contain a ? or other script coding, no longer cause most search engines to exclude them.
- Pages generated dynamically by other types of database software (e.g., Active Server Pages, Cold Fusion) can be indexed if there is a stable URL somewhere that search engine spiders can find. Once these were largely shunned by search engines. There are now many types of dynamically generated pages like these that are found in most general web search engines. There must a stable link to the page somewhere.

International Data Corporation predicts that the number of surface Web documents will grow from the current two billion or so to 13 billion within three years, a factor increase of 6.5 times; deep Web growth should exceed this rate, perhaps increasing about nine-fold over the same period. Fig. 8 compares this growth with trends in the cumulative global content of print information drawn from a recent UC Berkeley study.

A number of recent studies have noted that a tremendous amount of content on the Web is dynamic. This dynamism takes a number of different forms. For instance, web pages can be dynamically generated,
i.e., a server-side program creates a page after the request for the page is received from a client. Similarly, pages can be dynamic because they include code that executes on the client machine to retrieve content from remote servers (e.g., a page with an embedded applet that retrieves and displays the latest stock information).

Based on studies conducted in 1997, Lawrence and Giles estimated that close to 80% of the content on the Web is dynamically generated, and that this number is continuing to increase. As major software vendors come up with new technologies to make such dynamic page generation simpler and more efficient, this trend is likely to continue.

However, little of this dynamic content is being crawled and indexed. Current-day search and categorization services cover only a portion of the Web called the publicly indexable Web. This refers to the set of web pages reachable purely by following hypertext links, ignoring search forms and pages that require authorization or prior registration. In this paper, we address the problem of crawling a subset of the currently uncrawled dynamic Web content. In particular, we concentrate on extracting content from the portion of the Web that is hidden behind search forms in large searchable databases (the so called Hidden Web)

**EXTRACTION OF INFORMATION FROM HIDDEN WEB**

Crawlers are programs that automatically traverse the Web graph, retrieving pages and building a local repository of the portion of the Web that they visit. Depending on the application at hand, the pages in the repository are either used to build search indexes, or are subjected to various forms of analysis (e.g., text mining). Traditionally, crawlers have only targeted a portion of the Web called the publicly indexable Web (PIW). This refers to the set of pages reachable purely by following hypertext links, ignoring search forms and pages that require authorization or prior registration.

Large portions of the Web are ‘hidden’ behind search forms, in searchable structured and unstructured databases (called the hidden Web or deep Web). Pages in the hidden Web are dynamically generated in response to queries submitted via the search forms. The hidden Web continues to grow, as organizations with large amounts of high-quality information (e.g., the Census Bureau, Patents and Trademarks Office, news media companies) are placing their content online, providing Web-accessible search facilities over existing databases.

In this paper, we address the problem of building a hidden Web crawler; one that can crawl and extract content from these hidden databases. Such a crawler will enable indexing, analysis, and mining of hidden Web content, akin to what is currently being achieved with the PIW. In addition, the content extracted by such crawlers can be used to categorize and classify the hidden databases.

**A generic Hidden Web crawling algorithm**

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

Algorithm for Crawling a Hidden Web site

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

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

Problem formalization

Theoretically, the problem of query selection can be formalized as follows: it is assumed that the crawler downloads pages from a Web site that has a set of pages S (the rectangle in Fig. 2). Each Web page is represented in S as a point (dots in Fig. 2). Every potential query qi that may be issued can be viewed as a subset of S, containing all the points (pages) that are returned when we issue qi to the site. Each subset is associated with a weight that represents the cost of issuing the query. Under this formalization, the goal is to find which subsets (queries) cover the maximum number of points (Web pages) with the minimum total weight (cost). This problem is equivalent to the set covering problem in graph theory. There are two main difficulties that should be addressed in this formalization. First, in a practical situation, the crawler does not know which Web pages will be returned by which queries, so the subsets of S are not known in advance.

Without knowing these subsets the crawler cannot decide which queries to

![Fig. 2. A set-formalization of the optimal query selection problem](image-url)
pick to maximize the coverage. Second, the set-covering problem is known to be NP-Hard, so an efficient algorithm to solve this problem optimally in polynomial time has yet to be found. In this paper, a near optimal solution at a reasonable computational cost can be found with an approximation algorithm. The algorithm leverages the observation that although it is not known which pages will be returned by each query $q_i$ that is issued, it can be predicted how many pages will be returned. Based on this information the query selection algorithm can then select the “best” queries that cover the content of the Web site.

THE HIDDEN WEB CRAWLING

Conventional crawlers rely on the hyperlinks on the Web to discover pages, so current search engines cannot index the Hidden-Web pages (due to the lack of links). This is believed that an effective Hidden-Web crawler can have a great impact on how users search information on the Web. Given that the only “entry” to Hidden Web pages is through querying a search form, there are two core challenges to implementing an effective Hidden Web crawler:

1. The crawler has to be able to understand and model a query interface, and
2. The crawler has to come up with meaningful queries to issue to the query interface. Here, it presents a solution to the second challenge, i.e. how a crawler can automatically generate queries so that it can discover and download the Hidden Web pages. Clearly, when the search forms list all possible values for a query (e.g., through a drop-down list), the solution is straightforward and can issue all possible queries, one query at a time. When the query forms have a “free text” input, however, an infinite number of queries are possible, so it cannot issue all possible queries. Come up with meaningful queries without understanding the semantics of the search form.

The aim is to crawl the selective portion of the hidden Web, extracting contents based on the requirements of a particular application or task. For example, consider a market analyst who is interested in building an archive of news articles, reports, press releases, and white papers pertaining to the semiconductor industry, and dated sometime in the last ten years. There are two steps in building this archive: resource discovery, wherein sites are identified and databases that are likely to be relevant to the task; and content extraction, where the crawler actually visits the identified sites to submit queries and extract hidden pages. In this paper, the resource discovery is not directly addressed. Rather, the work examines how best to automate content retrieval, given the results of the resource discovery step.

Human-assistance is critical to ensure that the crawler issues queries that are relevant to the particular task. For instance, in the above example, the market analyst may provide the crawler with lists of companies or products that are of interest. This enables the crawler to use these values when filling out forms that require a company or product name to be provided. Furthermore, it can be seen that the crawler will be able to gather additional potential company and product names as it visits and processes a number of pages.

The fundamental difference between the actions of a hidden Web crawler, such as HiWE, and that of a traditional crawler is with respect to pages containing search forms. Fig. 3 illustrates the sequence of steps (as indicated by the numbers above each arrow) that take place, when a user uses a search form to submit queries on a hidden database. The user first get the form to be filled then the user fill-out the form and submit it to the web query front end.
which directly communicates with the hidden database. After submission the form response page come and the user can see the resultant page. Thus this is the user form interaction as shown in the Fig. 3. But our main aim is to fill these form by the software program on which research is going on.

Above figure illustrates the same interaction, with the crawler now playing the role of the human browser combination. The model of a hidden Web crawler consists of the four components described here (see Fig. 4). Form page is used to denote the page containing a search form, and response page, to denote the page received in response to a form submission.

**Internal Form Representation**

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

\[
F = \{(E_1; E_2; \ldots; E_n); S; M\},
\]

where \( E_1; E_2; \ldots; E_n \) is a set of \( n \) form elements, \( S \) is the submission information associated with the form (e.g., submission URL, internal identifiers for each form element, etc.), and \( M \) is meta-information about the form (e.g., URL of the form page, web-site hosting the form, set of pages pointing to this form page, other text on the page besides the form, etc.). A form element can be any one of the standard input elements: selection lists, text boxes, text areas, checkboxes, or radio buttons. For example, Fig. 6 shows a form with three elements (label \( (E_i) \) and dom \( (E_i) \) for now). Details about the actual contents of \( M \) and the information associated with each \( E_i \) are specific to a particular crawler implementation.

**Architecture of hidden web crawler**

A task-specific hidden Web crawler called the Hidden Web Exposer (HiWE) built at Stanford. Fig. 5 illustrates HiWE’s architecture and execution flow. Since search forms are the entry-points into the hidden Web, HiWE is designed to automatically process, analyze, and submit forms, using an internal model of forms and form submissions. This model treats forms as a set of (element, domain) pairs.

A form element can be any one of the standard input objects such as selection lists, text boxes or radio buttons. Each form
element is associated with a finite or infinite domain and a text label that semantically describes the element (Fig. 6). The values used to fill out forms are maintained in a special table called the LVS (Label Value Set) table (Fig. 5). Each entry in the LVS table consists of a label and an associated fuzzy/graded set of values (e.g., Label = "State" and value set = \{("Haryana", 0.8), ("Punjab", 0.7))\). The weight associated with a value represents the crawler’s estimate of how effective it would be, to assign that value to a form element with the corresponding label.

The basic actions of HiWE (fetching pages, parsing and extracting URLs, and adding the URLs to a URL list) are similar to those of traditional crawlers. However, whereas the latter ignore forms, HiWE performs the following sequence of actions for each form on a page:

The basic crawler data structure is the URL List. It contains all the URLs that the crawler has discovered so far.

The Crawl Manager controls the entire crawling process. In the implementation, the crawler was configured to operate within a predetermined set of target sites provided to the Crawl Manager at startup.

The Parser extracts hypertext links from the crawled pages and adds them to the URL List structure. Pages that do not contain forms are handled solely by the Parser and Crawl Manager modules.

The Form Analyzer, Form Processor, and Response Analyzer modules, together implement the form processing and submission operations of the crawler.

The LVS table is HiWE’s implementation of the task-specific database. The LVS Manager manages additions and accesses to the LVS table.

CONCLUSION

Traditional crawlers normally follow links on the Web to discover and download pages. Therefore they cannot get to the Hidden Web pages which are only accessible through query interfaces. In this paper, the study is made about the hidden web and all the related aspects for searching the hidden web and how the hidden websites can be downloaded for extraction of the information in the hidden databases. It is also studied that how a Hidden Web crawler can be built that can automatically query a Hidden Web site and download pages from it. The hidden web crawler can make a large number of queries for downloading the maximum possible data from the hidden web site. In particular, in certain cases the adaptive policy can download more than 90% of a Hidden Web site after issuing approximately 100 queries. Given these results, it is believed that the work provides a potential mechanism to improve the search-engine coverage of the Web and the user experience of Web search.

REFERENCES


5. Sriram Raghavan and Hector Garcia-Molina, “Crawling the hidden web”.


ABOUT THE AUTHORS

Rosy Madaan received B.E. degree in Computer Science & Engineering and M.Tech. in Computer Engineering with Hons. from Maharshi Dayanand University in 2005 and 2009 respectively. Presently, she is pursuing Ph.D. in Computer Engineering. Her areas of interests are Search Engines, Crawlers and Hidden Web.

Ashutosh Dixit received his Ph.D. (Computer Engg.) and M.Tech. (Computer Engg.) from MD University, Rohtak, Haryana in the year 2010 and 2004 respectively. Currently he is working as Associate Professor in Computer Engg. Deptt. at YMCA University of Science and Technology, Faridabad, Haryana. His research interest includes Data Structures, Computer Networks, Mobile Communications and Internet Technologies.

Prof. A.K. Sharma received his M.Tech. (Computer Sci. & Tech.) 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 Institute of Engineering, Faridabad in April 2002. He obtained his second Ph.D. in IT from IIIT & M, Gwalior in the year 2004. His research interest include Fuzzy Systems, Object Oriented Programming, Knowledge Representation and Internet Technologies.