PyBot: An Algorithm for Web Crawling

Alex Goh Kwang Leng
Department of Electrical and Computer Engineering
Curtin University
Miri, Malaysia
alexgoh.kwangleng@gmail.com

Ravi Kumar P
Department of Electrical and Computer Engineering
Curtin University
Miri, Malaysia
ravi2266@gmail.com

Ashutosh Kumar Singh
Department of Electrical and Computer Engineering
Curtin University
Miri, Malaysia
ashutosh.s@curtin.edu.my

Rajendra Kumar Dash
Department of Electronics & Comp. Sc.
K. S. U. B. College,
Bhanjanagar, India
rajbnj@rediffmail.com

***Abstract*—PyBot is a Web Crawler developed in Python to crawl the Web using Breadth First Search (BFS). The success of the World Wide Web (WWW), which itself built on the open internet, has changed the way how human share and exchange information and ideas. With the explosive growth of the dynamic Web, users have to spend much time just to retrieve a small portion of the information from the Web. The birth of the search engines have made human lives easier by simply taking them to the resources they want. Web crawler is a program used by search engines to retrieve information from the World Wide Web in an automated manner.**

***Keywords*– Crawler; Robots; Spiders; Robots Protocol; Crawler Policies**

I. Introduction

Web Crawlers are computer programs that traverse the World Wide Web in a systematic way with the intention of gathering data. The Robots doesn't actually move around to different computers on the Internet [1], as viruses or intelligent agents do. Each Robot opens many connections at once. This is necessary to retrieve web pages at a fast enough pace. A Robot resides on a single machine and sends HTTP requests for documents to other machines on the Internet, just as a web browser does when the user clicks on links. All the Web Crawlers really does is to automate the process of following links. Most of the Web Crawlers adopt a breadth-first retrieval strategy to increase the coverage of the Web. Our PyBot Crawler also uses the breadth-first retrieval strategy.

The main objective of a search engine is to provide more relevant results in a faster time over a rapidly expanding Web. There are three important sequential tasks a standard search engine does as shown in Figure 1. They are

- Crawling
- Indexing
- Searching

Among them, crawling is the most important one. A Web crawler is used for many purposes. The main purpose of the crawler is to download Web pages for indexing, and other purposes like page validation, structural analysis, visualization, update notification, mirroring and for the spam purposes like harvesting email addresses etc.

The indexing module provides information about ranking of pages back to crawlers so that the crawlers can collect the important pages first [24] as shown in Figure 1. Web crawlers are called by many names like spiders, robots, bots and ants.

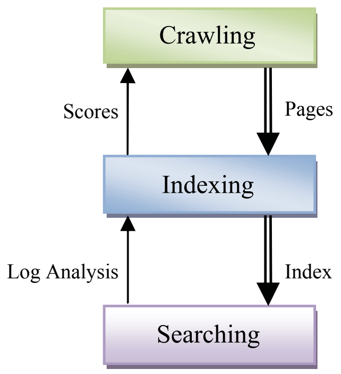


Figure 1. General sequential task of a search engine

The main purpose of the crawler is to visit pages in the Web and download them in a systematic way for the search engines. A crawler starts with a universal resource locator (URL), explores all the hyperlinks in that page, visit those pages and download the pages. These downloaded pages are indexed and stored for search engines. A search engine will be rated based on its search performance, quality of the results, and its ability to crawl and index the Web efficiently. That is why search engines are doing a lot of research on making a better Web crawler.

Web crawlers are developed for doing a variety of jobs. The following jobs are important ones done by a crawler. They are:

- Download the Web page
- Parse through the downloaded pages and retrieve all the hyperlinks
- For each link retrieved, repeat the process

A crawler can be used for crawling through a whole site on the internet or intranet; it works well for our PyBot. Some crawlers uses parallel download in order to reduce time. Once a URL is specified, the crawler follows all the links found in that page and go to all the links and download the pages. It continues until all the links are explored or the crawler wants to stop. A crawler treats a Web site as a tree-structure as shown in Figure 2. Here the start URL is the root of the tree and the children of the root are their links. Some children may not have any links like the child 2 of root and they are called as dangling child.

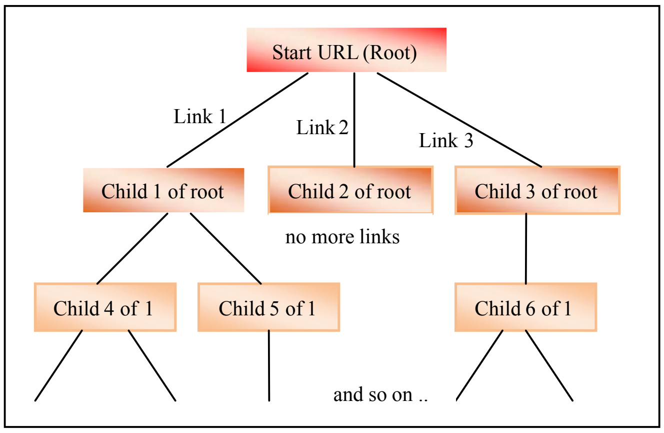


Figure 2. Web Site as a Tree-Structure

Developing and running a Web crawler is a challenging task because it involves performance, reliability and social issues [1]. Performance issue is related with the speed of the Web crawler discovering the Web, reliability issue is related with how the crawler is able to interact with various Web servers and name servers during crawling. Social issue is related with how the crawler respects the Web site policies like not to visit some pages etc. There are also other factors which affect the performance of the crawlers like Web site policies, explosive growth of internet, rate of changes, growth of dynamic pages and the storage requirements.

This paper is organized as follows. Next Section shows previous work on the crawlers. Section III provides our proposed method for the Crawlers. Section IV provides our experimental results of the proposed method. This paper is concluded in Section V with future work.

II. Previous Work

There are a few crawling techniques used by Web Crawlers, the important ones are highlighted here, which are: *General Purpose Crawling*: A general purpose Web Crawler gathers as many pages as it can from a particular set of URL's and their links. In this, the crawler is able to fetch a large number of pages from different locations. General purpose crawling can slow down the speed and network bandwidth because it is fetching all the pages. *Focused Crawling*: A focused crawler is designed to gather documents only on a specific topic, thus reducing the amount of network traffic and downloads. The goal of the focused crawler is to selectively seek out pages that are relevant to a pre-defined set of topics. It crawl only the relevant regions of the web and leads to significant savings in hardware and network resources. The most crucial evaluation of focused crawling is to measure the harvest ratio, which is rate at which relevant pages are acquired and irrelevant pages are effectively filtered off from the crawl. This harvest ratio must be high, otherwise the focused crawler would spend a lot of time merely eliminating irrelevant pages, and it may be better to use an ordinary crawler instead [3]. *Distributed Crawling:* In distributed crawling, multiple processes are used to crawl and download pages from the Web thus building a scalable, easily configurable system, which is fault tolerant system. Splitting the load decreases hardware requirements and at the same time increases the overall download speed and reliability [3].

A. *Robots Exclusion Standards*

In the Web Crawling World, there are some rules that Web Crawlers should follow, these rules are called Robot Policies or Robots Exclusion Protocol or even robots.txt protocol. It is a convention to prevent cooperating Web Crawlers from accessing all or part of a website. The following are some policies that a Robot must follow when it is crawling the Web:

- All the Robots must access the robots.txt file before downloading any files
- Robots must declare their identity to the Web servers
- Robots must minimize the burden of Web servers by using a low retrieval rate and access the Web server only when the server is lightly loaded

A site owner can wish to give instructions to web crawlers by placing a text call robots.txt in the root of the web site hierarchy (e.g. www.curtin.edu.my/robots.txt). This text file should contain the instructions in a specific format. A robots.txt file on a website will function as a request that specified robots ignored specified files or directories in their search. Poorly designed Robots can lead to serious network and server overload problems that is why all Robots must follow the Robot Policies. Also, Robots that did not follow the Robot Policies might get blacklisted or banned from the website.

B. *Search Engine types*

There are 5 main types of Search Engine. They are:

1) *Crawler-Based Search Engines:* Spider or Crawler based search engine uses an automated program called Spider or robot or crawler or bot to crawl the web pages, download and create index and store in their database. When a user perform a search, the search engine will check its database of web pages for the keyword the user searched on to present a list of link results. They contain a full text of the web pages they link to. Spider-Based search engines are constantly searching the internet updating their database of information with the new or altered pages. Examples are Google and Yahoo! Search search engines.

2) *Directory Based Search Engines:* A directory-based search engine uses human editors to edit which category the page belongs and place the page in that category in the 'directories' database. Example is Yahoo! directory search engine.

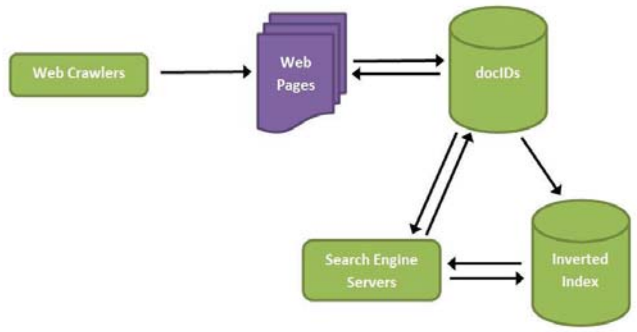
3) *Meta Search Engines:* Meta-search engines (also known as multi-threaded engines) search several major engines at once. Meta-search engines do not crawl the Web or maintain a database of web pages. Instead, they act as a middle agent, passing on the query to the major engines and then returning the results. Examples are Dogpile and Vivisimo search engines.

4) *Hybrid Search Engines:* Hybrid search engine uses a combination of both crawler-based results and human-powered directory results. For example, MSN Search is more likely to present human-powered listings from LookSmart.

However, it does also present crawler-based results provided by Inktomi.

5) Concept Based Search Engines: Concept-based search systems try to determine what you mean, not just what you say. In the best circumstances, a concept-based search returns hits on documents that are related to the subject or theme that explores, even if the words in the document do not precisely match the words you enter into the query. Example Excite, Essie (for structured medical text), Compass (for HTML, deep web data).

A standard Web Search Engine Architecture is shown in Figure 3. Standard search engine architecture consists of three main parts – web crawling, indexing and queries. The queries are done after these three main components finished first. It starts with a bunch of machines that act as crawlers or spiders or sometimes called robots which are programmed to crawl the Web. They go out and make HTTP request all over the Web and try to gather Web pages. After that they stored the Web pages in a database. Duplicated Web pages are checked and not stored in the database. Once the crawlers done their crawling, the indexer create an inverted index table for the data they collected. After that, inverted index are sent to search engine servers and waiting for query to process. So a user might went to a search engine website and submit a query to the search engine servers and the search engine servers can immediately sent the results back to the user.



F

igure 3. Standard Web Search Engine Architecture

C. Web Crawler Policies

The outcome of a Web Crawler is normally evaluated based on how they adhered to the following four policies. They are:

1) Selection Policy: Selection policy refers to the important pages needs to be downloaded first for efficiency purposes considering the bulk amount of data on the Web. According to Lawrence and Giles [4], that no search engine covers more than 16% of the Web contents. A recent study by Gulli et al. [5] showed that a large-scale search engines index no more than 40%-70% of the indexable Web. Web crawler is highly desirable to downloaded the most relevant pages, and not just a random sample of the Web. The importance of a page is a function of its intrinsic quality, its popularity in terms of links or visits, and even of its URL. The selection policy needs to be revised after crawling in to the actual Web to improve its quality of pages it retrieves. Cho et al. [6] made the first study on policies for Crawler scheduling within the Stanford.edu domain with different strategies like Breadth-First, Backlink-Count and Partial Pagerank. Najork and Wiener [7] performed an actual crawl on 328 million pages, using breadth-first ordering. Abiteboul et al. [8] and Boldi et al. [9] did research and simulation on selection policy with different strategies and among the results the Breadth-First strategy works better. Chakrabarti et al. [10] introduced focused crawling. H.T. Lee et al. [11] did research on scaling to 6 billion pages crawling using a single server and compared their results.

2) Re-visit Policy: Most of the Web sites are now dynamic in nature, and crawling a Web can take a long time usually in weeks or months. By the time the Web crawler has finished its crawl, many things could have happened like new contents might have added, contents might have updated and some contents might have deleted [12]. From the search engine's point of view, there is a cost associated with not detecting an event, and thus having an outdated copy of a resource. According to J. Cho et al. [13] the most used cost functions are freshness and age. Two simple re-visiting policies were studied by Cho and Garcia-Molina [14]. The first one is *Uniform policy*, this involves re-visiting all pages in the collection with the same frequency, regardless of their rates of change. The second one is *Proportional policy*, this involves re-visiting more often the pages that change more frequently. The visiting frequency is directly proportional to the (estimated) change frequency.

3) Politeness Policy : According to M. Koster [15], the use of Web Crawler is useful for a number of tasks, but comes with a price. The costs of using Web Crawler include network resources, server overload, poorly written robots and also Personal robots that disturb the networks and servers.

Cho et al. [16] suggests 10 seconds as an interval for in their studies. Anecdotal evidence from access logs shows that access intervals from known crawlers vary between 20 seconds and 3–4 minutes.

4) Parallelization policy : To increase the performances of the Web crawler, especially crawling on large data repository like World Wide Web, often they include technologies like parallel computing for the crawlers. A Web crawler can be an agent and by executing multiple agents, with interaction between the agents, the Web can be crawled faster and more efficient. Another way is that there is a crawling system that assigned URL to its crawlers so that they does not crawl the same URL.

D. Existing Web Crawlers

There are a lot works and research done on Web Crawlers. Web Crawlers are developed for research and commercial purpose. WebBase [2] is one of the crawler developed by the Stanford University for research purpose. The WebBase is the continuation of the original Google repository and search engine at the Stanford University. The following are the list of general-purpose crawlers:

- **Yahoo! Slurp** : It is the Yahoo Search Engine's Crawler
- **Msnbot**: It is the name of Microsoft's Bing web crawler
- **Googlebot** [1]: It is the Crawler used by Google, which collects documents from the web to build a searchable index for the Google search engine
- **Methabot**: is a scriptable web crawler written in C, released under the ISC license

- **PolyBot** [17]: It is a distributed crawler written in C++ and Python, which is composed of a "crawl manager", one or more "downloaders" and one or more "DNS resolvers"
- **FAST Crawler** [18]: It is a distributed crawler, used by Fast Search & Transfer, and a general description of its architecture is available.
- **RBSE** [19]: It was the first published web crawler. It was based on two programs: the first program, "spider" maintains a queue in a relational database, and the second program "mite", is a modified www ASCII browser that downloads the pages from the Web.
- **WebCrawler** [20]: It was used to build the first publicly-available full-text index of a subset of the Web. It was based on lib-WWW to download pages, and another program to parse and order URLs for breadth-first exploration of the Web graph. It also included a real-time crawler that followed links based on the similarity of the anchor text with the provided query.
- **World Wide Web Worm** [21]: It was a crawler used to build a simple index of document titles and URLs. The index could be searched by using the grep, UNIX command.
- **WebFountain** [22]: It is a distributed, modular crawler similar to Mercator but written in C++.
- **WebRACE** [23]: It is a crawling and caching module implemented in Java, and used as a part of a more generic system called eRACE.

There are also some popular open-source Crawlers like Aspseek, crawler4j, GRUB, WIRE [24] and many more on the list.

III. PROPOSED ALGORITHM

Our method uses the standard Breadth-First Search strategy to design and develop a Web Crawler called PyBot. The High-level Web Search Engine Architecture used for this PyBot Crawler is shown in Figure 4.

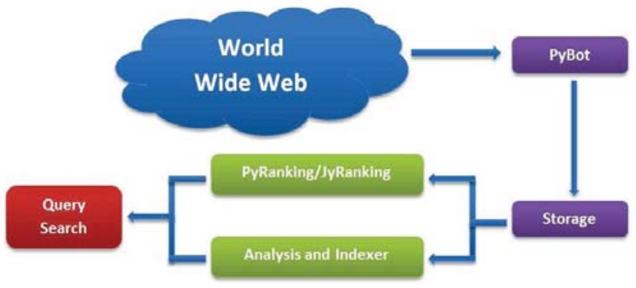


Figure 4. High-level Web Search Engine Architecture

PyBot is a simple Web Crawler written in Python 2.7. Initially it takes an URL and from that URL, it gets all the hyperlinks. From the hyperlinks, it crawls again until a point that no new hyperlinks are found. It downloads all the Web Pages while it is crawling. PyBot will output a Web structure in Excel CSV format on the website it crawls. Both downloaded pages and Web structure in Excel CSV format are stored in storage and are used for the PyRanking/JyRanking and Analysis and Indexer. PyRanking and JyRanking are the ranking system used in this method. The ranking systems take the Web structure in Excel CSV format and apply the PageRank algorithm and produces ranking order of the pages by displaying the page list with most popular pages at the top.

In our proposed method, the analysis and indexer and the query search modules are not implemented due to limited resources.

We used the standard and the most popular BFS search algorithm for crawling the Web. BFS uses the strategy in which the root node is expanded first, and then all the successors of the root node are expanded next, then their successors, and so on. In general, all the nodes are expanded at a given depth in the search tree before any nodes at the next level are expanded. The same logic is applied to the Web site during crawling. The root node is the URL and all the hyperlinks are visited and downloaded and so on. A sample 8 node Web Graph is shown in Figure 5. Figure 6 shows the order of nodes crawled using BFS method using a tree and Figure 7 shows using a queue. The Table I shows the outgoing links of all the nodes.

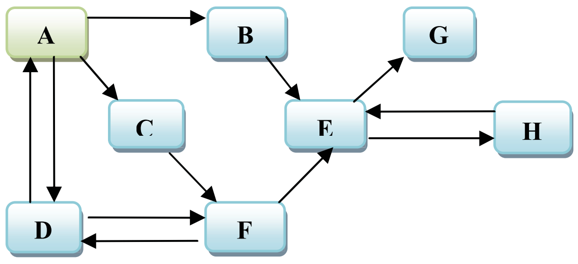


Figure 5. A Sample 8 Node Web Graph

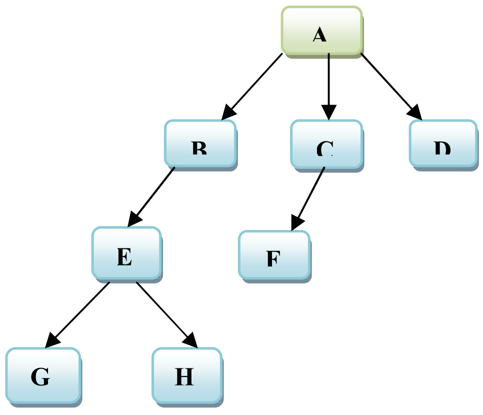


Figure 6. Order of Nodes Crawled Using BFS Shown Using a Tree

A	B	C	D	E	F	G	H

Figure 7. Order of Nodes Crawled Using BFS Shown Using a Queue

TABLE I. OUTGOING LINKS FOR EACH NODE

Pages	Outgoing Pages		
A	B	C	D
B	E		
C	F		
D	A	F	

E	G	H	
F	D	E	
G			
H	E		

Our PyBot Crawler is implemented using Tree-Search along with a First-In-First-Out (FIFO) Queue to assure that the pages that are visited first will be expanded first. The FIFO queue puts all newly generated links at the end of the queue, which means that shallow pages are expanded before deeper pages. Figure 7 shows the order of nodes crawled using a Queue. The BFS algorithm is shown below.

Algorithm : Breadth First Search Crawling
Require : Starting URL, $Q = \{p_{start}\}$, queue of URLs to visit

```
1.  If $p_{start}$ violated the Robot Exclusion Protocol, then
     Exit
2.  $V = \emptyset$, visited URLs
3.  while $Q \neq \emptyset$ do
4.      Dequeue $p \in Q$
5.      Parse $p$ to extract text and outgoing links
6.      $\Gamma^{+}(p) \leftarrow$ all the hyperlinks inside $p$
7.      for each $p' \in \Gamma^{+}(p)$ do
8.          if $p' \notin V \wedge p' \notin Q$ then
9.      if not violated the Robot Exclusion Protocol then
10.         $Q = Q \cup \{p'\}$
11.     end if
12. end if
13.      end for
14. end while
```

IV. Experimental Results

PyBot Crawler is tested on the 5th November 2010 in the Curtin Website (http://www.curtin.edu.my/), PyBot crawls the site and retrieve all the hyperlinks and save it in an Excel CSV format. Along with that it also saves additional information like all the crawled pages, dangling pages, non-dangling pages, visited robot pages, robots.txt pages and external outgoing links. These pages are saved as text files. PyBot neglects external outgoing links so that it will not keep crawling the external pages. While it crawls, it downloads the pages at the same time.

PyRanking takes the downloaded pages and apply the PageRank algorithm and produces the most popular pages on the top.

The programs are tested on an Intel Core 2 Duo E8400 (3.00 GHz) with 3 GB RAM. The screen shot of the input screen used to type the URL is shown in Figure 8. The screen shot of the PyBot crawling is shown in Figure 9.

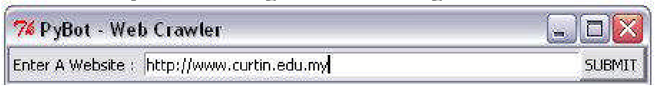
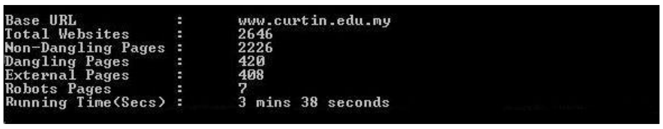


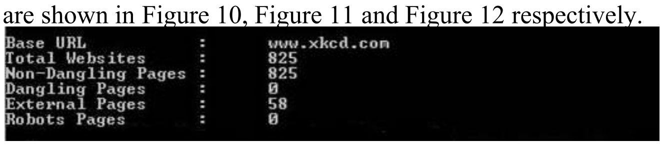
Figure 8. Input Screen Used To Type URL



```
Base URL            :      www.curtin.edu.my
Total Websites      :      2646
Non-Dangling Pages  :      2226
Dangling Pages      :      420
External Pages      :      408
Robots Pages        :      7
Running Time(Secs)  :      3 mins 38 seconds
```

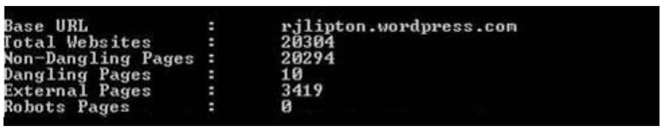
Figure 9. PyBot Crawling Screen

Figure 9 shows the PyBot crawling screen, the URL crawled (in our case we used our URL, www.curtin.edu.my), total pages crawled, non-dangling pages, dangling pages, external pages, robot pages and the total time to crawl. The PyBot crawler is also run on other sites like XKCD, W3Schools and Tutorialspoint. The screen shots of those sites are shown in Figure 10, Figure 11 and Figure 12 respectively.



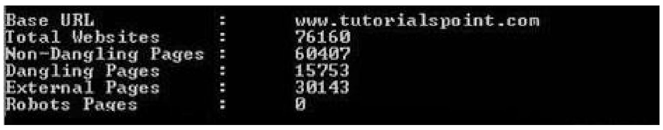
```
Base URL            :      www.xkcd.com
Total Websites      :      825
Non-Dangling Pages  :      825
Dangling Pages      :      0
External Pages      :      58
Robots Pages        :      0
```

Figure 10. Crawler Summary for XKCD Site



```
Base URL            :      rjlipton.wordpress.com
Total Websites      :      20304
Non-Dangling Pages  :      20294
Dangling Pages      :      10
External Pages      :      3419
Robots Pages        :      0
```

Figure 11. Crawler Summary for rjlipton's blog



```
Base URL            :      www.tutorialspoint.com
Total Websites      :      76160
Non-Dangling Pages  :      60407
Dangling Pages      :      15753
External Pages      :      30143
Robots Pages        :      0
```

Figure 12. Crawler Summary for Tutorialspoint

V. Conclusion

This paper focuses on the basic concepts of a Web crawler and provides an algorithm for Web Crawling. To start our research, we implemented a simple BFS based Crawler called PyBot and crawled in our University site and collected the data. The crawler also crawled few more sites, collected data like total pages in a site, the number of non-dangling pages, dangling pages, external pages, robots pages and the crawling time and compared. We applied the PageRank algorithm to the pages we collected using our Crawler and that is not discussed here because it is beyond the scope of this paper.

One possible future work will be research on the indexing and analysis area and include those modules in the PyBot crawler and make a full fledged Web Crawler. Another area will be to make a better ranking algorithm based on link structure analysis.

References

[1] S. Brin, and L. Page, "The anatomy of a large-scale hypertextual Web search engine". Proc. of the World Wide Web Conference (WWW'98), pp 107-117, 1998.

[2] J. Cho, H. Garcia-Molina, T. Haveliwala, W. Lam, "Stanford WebBase Components and Applications", ACM Transactions on Internet Technology, Vol. 6, No. 2, pp. 153–186, 2006.

[3] P. Baldi, P. Frasconi, and P. Smyth, "Modeling the Internet and the Web: Probabilistic Methods and Algorithms", John Wiley & Sons Publishers, 2003.

[4] S. Lawrence and C. Lee Giles, "Accessibility of information on the web", Nature, 400(6740):107–109, 1999.

[5] A. Gulli and A. Signorini, "The Indexable Web is More than 11.5 Billion Pages", Proc. of the World Wide Web Conference (WWW05), pp 902-903, 2005.

[6] J. Cho, H. Garcia-Molina and L. Page, Efficient crawling through URL ordering. In Proceedings of the seventh conference on World Wide Web, Brisbane, Australia, April 1998.

[7] M. Najork and J. L. Wiener. "Breadth-first crawling yields high-quality pages", Proc. of the Tenth Conference on World Wide Web, pp 114–118, Hong Kong, May 2001.

[8] S.Abiteboul, M. Preda, and G. Cobena, "Adaptive on-line page importance computation", Proc. of the twelfth international conference on World Wide Web, pp, 280–290., 2003.

[9] P. Boldi, M. Santini, and S. Vigna, "Do your worst to make the best: Paradoxical effects in pagerank incremental computations", Proc of the third Workshop on Web Graphs (WAW*)*, volume 3243 of Lecture Notes in Computer Science, pages 168–180, Rome, Italy, October 2004

[10] S. Chakrabarti, M.Van den Berg, and B. Dom, "Focused crawling: a new approach to topic-specific web resource discovery", Computer Networks, 31(11–16):1623–1640, 1999.

[11] H.T. Lee, D. Leonard, X. Wang and D. Loguinov, „IRLbot: Scaling to 6 Billion Pages and Beyond", ACM Transactions on the Web, Vol. 3, No 3, Article 8, June 2009.

[12] R. Baeza-Yates, C. Castillo, and F. Saint-Jean. Web Dynamics, chapter Web Dynamics, Structure and Page Quality, pages 93–109. Springer, 2004.

[13] J. Cho and H. Garcia-Molina, "Synchronizing a database to improve freshness", Proc. of ACM International Conference on Management of Data (SIGMOD), pp 117–128, Dallas, Texas, USA, May 2000.

[14] J. Cho and H. Garcia-Molina, "Effective page refresh policies for web crawlers", ACM Transactions on Database Systems, 28(4), December 2003.

[15] Martijn Koster. Robots in the web: threat or treat ? *ConneXions*, 9(4), April 1995.

[16] J. Cho and H. Garcia-Molina, "Estimating frequency of change", ACM Transactions on Internet Technology, 3(3), August 2003.

[17] V. Shkapenyuk and T. Suel, "Design and implementation of a high performance distributed web crawler", Proc. of the 18th International Conference on Data Engineering (ICDE), pp 357-368, San Jose, California. IEEE CS Press, 2002.

[18] K.M. Risvik and R. Michelsen, R, "Search Engines and Web Dynamics", Computer Networks, vol. 39, pp. 289–302, June 2002.

[19] D. Eichmann, "The RBSE spider: balancing effective search against Web load", *Proc. of the First World Wide Web Conference,* Geneva, Switzerland, 1994.

[20] B. Pinkerton, "Finding what people want: Experiences with the WebCrawler", In Proc. of the First World Wide Web Conference, Geneva, Switzerland, 1994.

[21] O.A. McBryan, "GENVL and WWWW: Tools for taming the web", Proc. of the First World Wide Web Conference, Geneva, Switzerland, 1994.

[22] J. Edwards, K.S. McCurley and J.A. Tomlin, "An adaptive model for optimizing performance of an incremental web crawler". Proc. of the Tenth Conference on World Wide Web (Hong Kong: Elsevier Science): pp.106–113, 2001.

[23] D. Zeinalipour-Yazti and M.D. Dikaiakos, "Design and implementation of a distributed crawler and filtering processor.", Proc of the Fifth Next Generation Information Technologies and Systems (NGITS), volume 2382 of Lecture Notes in Computer Science, pp 58–74, Caesarea, Israel. Springer, 2002.

[24] Carlos Castillo, "Effective Web Crawling", Ph.D. dissertation, Dept. of Computer Science, University of Chile, 2004.