Application of A Productive Web Crawling Algorithm

Piyush Kumar[1], Kirti Bhatia[2]
[1]Dept of Computer Science and Engineering
[2]Assistant Professor, Dept of Computer Science and Engineering
[1, 2] SKITM, Bahadurgarh, India

Abstract- *By crawling, we mean to traverse the Web by recursively following links from a starting URL or a set of starting URLs. This starting URL set is the entry point though which any crawler starts searching procedure. This set of starting URL is known as "Seed Page". The selection of a good seed is the most important factor in any crawling process.Without search engines, it would be almost impossible for us to locate anything on the Web unless or until we know a specific URL address. Every search engine maintains a central repository or databases of HTML documents in indexed form. Whenever a user query comes, searching is performed within that database of indexed web pages. The size of repository of every search engine can't accommodate each and every page available on the WWW. So it is desired that only the most relevant pages are stored in the database so as to increase the efficiency of search engines. To store most relevant pages from the World Wide Web, a suitable and better approach has to be followed by the search engines. This database of HTML documents is maintained by special software. The software that traverses web for capturing pages is called "Crawlers" or "Spiders".In this paper, we implement the efficient web crawling algorithmto resolve the issues faced in earlier approaches or techniques.*

I. INTRODUCTION

As discussed, web crawler traverses the web and behind it there is an algorithm. In this paper, we have implemented an algorithm and shown the results that shows the performance of the algorithm. A web-crawler is a program/software or automated script which browses the World Wide Web in a methodical, automated manner.The structure of the World Wide Web is a graphical structure, *i.e.*, the links given in a page can be used to open other web pages. Actually, Internet is a directed graph, webpage as node and hyperlink as edge, so the search operation could be abstracted as a process of traversing directed graph. By following the linked structure of the Web, we can traverse a number of new web-pages starting from a Starting webpage. Web crawlers are the programs or software that uses the graphical structure of the Web to move from page to page [1]. Such programs are also called wanderers, robots, spiders, and worms. Web crawlers are designed to retrieve Web pages and add them or their representations to local repository/databases. Web crawlers are mainly used to create a copy of all the visited pages for later processing by a search engine that will index the downloaded pages that will help in fast searches. Web search engines work by storing information about many web pages, which they retrieve from the WWW itself. These pages are retrieved by a Web crawler (sometimes also known as a spider) — which is an automated Web browser that follows every link it sees. Web *crawlers* are programs that exploit the graph structure of the web to move from page to page. It may be observed that 'crawlers' itself doesn't indicate speed of these programs, as they can be considerably fast working programs.Web crawlers are software systems that use the text and links on web pages to create search indexes of the pages, using the HTML links to follow or crawl the connections between pages.

II. LITERATURE SURVEY

Web crawlers are almost as old as the web itself. The first crawler, Matthew Gray's Wanderer, was written in the spring of 1993, roughly coinciding with the first release of NCSA Mosaic. Several papers about web crawling were presented at the first two World Wide Web conferences. However, at the time, the web was three to four orders of magnitude smaller than it is today, so those systems did not address the scaling problems inherent in a crawl of today's web.

Obviously, all of the popular search engines use crawlers that must scale up to substantial portions of the web. However, due to the competitive nature of the search engine business, the designs of these crawlers have not been publicly described. There are two notable exceptions: the Google crawler and the Internet Archive crawler. The original Google crawler [2] (developed at Stanford) consisted of five functional components running in different processes. A *URL server process* read URLs out of a file and forwarded them to multiple crawler processes. Each *crawler process* ran on a different machine, was single-threaded, and used asynchronous I/O to fetch data from up to 300 web servers in parallel [3]. The crawlers transmitted downloaded pages to a single *StoreServer process*, which compressed the pages and stored them to disk. The pages were then read back from disk by an *indexer process*, which extracted links from HTML pages and saved them to a different disk file. A *URL resolver process* read the link file, derelativized the URLs contained therein, and saved the absolute URLs to the disk file that was read by the URL server [4]. Typically, three to four crawler machines were used, so the entire system required between four and eight machines [5].

Research on web crawling continues at Stanford even after Google has been transformed into a commercial effort. The Stanford WebBase project has implemented a high performance distributed crawler, capable of downloading 50 to 100 documents per second .Cho and others have also developed models of document update frequencies to inform the download schedule of incremental crawlers.

The Internet Archive also used multiple machines to crawl the web. Each crawler process was assigned up to 64 sites to crawl, and no site was assigned to more than one crawler. Each single-threaded crawler process read a list of seed URLs for its assigned sites from disk into per-site queues, and then used asynchronous I/O to fetch pages from these queues in parallel. Once a page was downloaded, the crawler extracted the links contained in it [6]. If a link referred to the site of the page it was contained in, it was added to the appropriate site queue; otherwise it was logged to disk. Periodically, a batch process merged these logged “cross-site” URLs into the site-specific seed sets, filtering out duplicates in the process [7].

III. WEB CRAWLING ALGORITHM

By looking at the basic working of crawlers, it is clear to us that at each stage crawler selects a new link from the frontier for processing. So the selection of the next link from the frontier is also a major aspect. The selection of the next link from the frontier entirely depends upon the crawling algorithm we are using.

So the job of selecting the next link from the frontier is something like selection of job by the CPU from Ready Queue (CPU Scheduling) in operating systems.The basic algorithm executed by any web crawler takes a list of ***seed* URLs** as its input and repeatedly executes the following steps:

- **Remove a URL from the URL list**
- **Determine the IP address of its host name**
- **Download the corresponding document**
- **Extract any links contained in it**

For each of the extracted links, ensure that it is an absolute URL and add it to the list of URLs to download; provided it has not been encountered before. If desired, process the downloaded document in other ways (e.g., index its content).

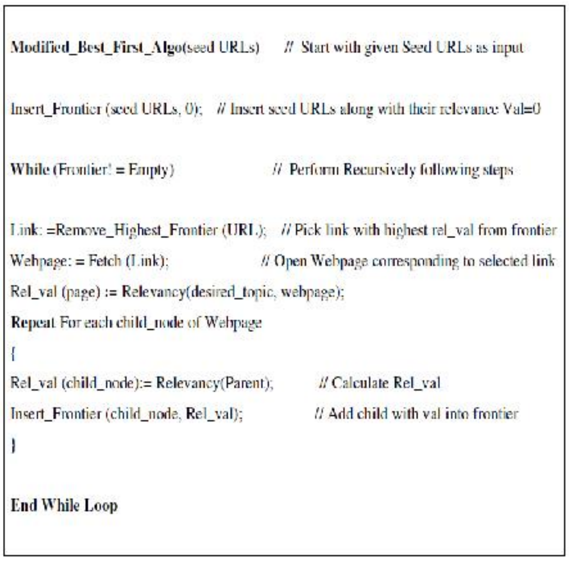
Any basic Web crawling algorithm requires a number of functional components:

- A component (called the URL frontier) for storing the list of URLs to download;
- A component for resolving host names into IP addresses;
- A component for downloading documents using the HTTP protocol;
- A component for extracting links from HTML documents; and
- A component for determining whether a URL has been encountered before or not.

The simplest crawling algorithm uses a queue of URLs yet to be visited and a fast mechanism for determining if it has already seen a URL. This requires huge data structures. A simple list of 20 billion URLs contains more than a terabyte of data. [How things] The crawler initializes the queue with one or more “seed” URLs. A good seed URL will link to many high-quality Web sites—for example, www.dmoz.org or wikipedia.org. Crawling proceeds by making an HTTP request to fetch the page at the first URL in the queue. When the crawler fetches the page, it scans the contents for links to other URLs and adds each previously unseen URL to the queue. Finally, the crawler saves the page content for indexing. Crawling continues until the queue is empty.

Modified Best First Algorithm

The Pseudo-code of Modified Best First approach can be stated as below:



```
Modified_Best_First_Algo(seed URLs)      // Start with given Seed URLs as input

Insert_Frontier (seed URLs, 0);   // Insert seed URLs along with their relevance Val=0

While (Frontier! = Empty)                 // Perform Recursively following steps

Link: =Remove_Highest_Frontier (URL);   // Pick link with highest rel_val from frontier
Webpage: = Fetch (Link);                 // Open Webpage corresponding to selected link
Rel_val (page) := Relevancy(desired_topic, webpage);
Repeat For each child_node of Webpage
{
Rel_val (child_node):= Relevancy(Parent);        // Calculate Rel_val
Insert_Frontier (child_node, Rel_val);           // Add child with val into frontier
}

End While Loop
```

Figure 6.1: Pseudo-code of modified Best Approach

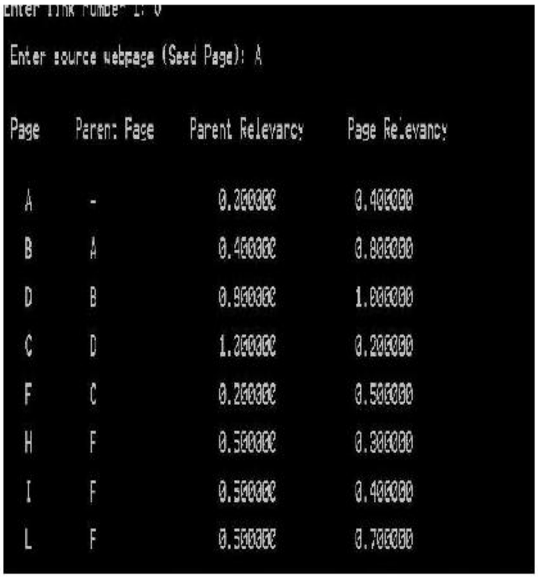
As cleared from the pseudo-code above, the initial SEED URLs are assigned relevancy value 0 and are inserted into the Frontier. Until the Frontier gets empty, the URL with highest relevancy value is fetched. Each URL is traversed and checked for relevancy.

The relevancy score of the node is assigned to each of its child nodes. Along with respective relevancy values inherited from parent node, new children URLs are inserted into the Frontier.

Results for Modified Best First Search

In our modified Best First Approach, the Relevancy Score of the Parent Page is also associated with the current page.

The output screen of the results is as shown below:



```
Enter link number 1: 0

Enter source webpage (Seed Page): A

Page     Parent Page     Parent Relevancy     Page Relevancy

 A         -                0.000000             0.400000
 B         A                0.400000             0.800000
 D         B                0.800000             1.000000
 C         D                1.000000             0.200000
 F         C                0.200000             0.500000
 H         F                0.500000             0.300000
 I         F                0.500000             0.400000
 L         F                0.500000             0.700000
```

Output Results for Modified Best First approach

As cleared from the output screenshot, the parent relevance decides which page will be selected next. The page with the highest parent relevance value is selected from the Frontier every time.We start with A. since it is the starting node, its parent is nothing and parent relevancy is also 0. Its own relevancy comes out to be 0.4.

Page A has 2 children – B and C. So both B and C are assigned parent relevance 0.4.(Note that the parent relevancy calculation is dynamic. i.e. if a page is traversed again from another parent, its parent value will depend on that new parent page and corresponding parent relevance will also change dynamically)

As here, C is approached again from Page D, since the relevancy of D comes out to be 1.0, t**he parent relevancy of C changes from 0.4 to 1.0.**Thus our modified program considers the dynamic nature of the web where contents of the pages could change anytime and thus the relevancy also gets affected.

IV. CONCLUSION & FUTURE WORK

Here we have implemented the modified algorithm for web crawling and can easily identify that it has provided best results. Internet is one of the easiest sources available in present days for searching and accessing any sort of data from the entire world. The structure of the World Wide Web is a graphical structure, and the links given in a page can be used to open other web pages. In this thesis, we have used the graphical structure to process certain traversing algorithms used in the search engines by the Crawlers. Each webpage can be considered as node and hyperlink as edge, so the search operation could be abstracted as a process of traversing directed graph. By following the linked structure of the Web, we can traverse a number of new web-pages starting from a Starting webpage. Web crawlers are the programs or software that uses the graphical structure of the Web to move from page to page. In this thesis, we have briefly discussed about Internet, Search Engines, Crawlers and then Crawling Algorithms.

There are number of crawling strategies used by various search engines. The basic crawling approach uses simple Breadth First method of graph traversing. But there are certain disadvantages of BFS since it is a blind traversing approach. To make traversing more relevant and fast, some heuristic approaches are followed. The results of all the crawling approaches are giving different results.

These heuristic searches keep a check on the relevancy factor of every page to be traversed. Thus the efficiency of the database of the search engine increases and only relevant pages are stored in it.The fast and more accurate version of Fish Search is known as – Shark Search, which is probably the best crawling approach. But there are some issues related with the implementation of Shark Search Approach in simple programming environment as there is lot of factors, calculations associated with it.

In future, work can be done to improve the efficiency of algorithms and accuracy and timeliness of the search engines. The work of Shark Search can be extended further to make Web crawling much faster and more accurate.

REFERENCES

[1] Junghoo Cho, Hector Garcia-Molina: “Parallel Crawlers**”,** 7–11 May 2021, Honolulu,Hawaii, USA.

[2] Articles about Web Crawlers available at Http://en.wikipedia.org/wiki/Web_crawler#Examples_of_Web_crawlers

[3] Marc Najork, Janet L. Wiener,” Breadth-first search crawling yields high-quality pages”, WWW10 proceedings in May 2-5, 2021, Hong Kong.

[4] Sergey Brin and Lawrence Page, “The Anatomy of a Large-Scale Hypertextual Web Search Engine”, Computer Science Department, Stanford University, Stanford, CA Available at- http://www.his.se/upload/51108/google.pdf

[5] Filippo Menczer, Gautam Pant and Padmini Srinivasan, “Topical Web Crawlers: Evaluating Adaptive Algorithms” ACM Transactions on Internet Technology, Vol. 4, No. 4, November 2019, Pages 378–419. Available at- http://www.informatics.indiana.edu/fil/Papers/TOIT.pdf

[6] P. De Bra, G. Houben, Y. Kornatzky and R. Post, "Information Retrieval in Distributed Hypertexts", in Proceedings of the 4th RIAO Conference, 481 - 491, New York, 2017.

[7] M. Hersovici, M. Jacovi, Y. Maarek, D. Pelleg, M. Shtalhaim and S. Ur "The Shark-Search Algorithm – An Application: Tailored Web Site Mapping", In *Proceedings of the Seventh International World Wide Web Conference*, Brisbane, Australia, April 2018. Available at- http://www7.scu.edu.au/1849/com1849.htm

[8] Sriram Raghavan, Hector Garcia-Molina, “Representing Web Graphs”, Stanford University, CA – June 2020. Available at- http://www.almaden.ibm.com/cs/people/rsriram/pubs/icde03.pdf

[9] A. Z. Broder, S. R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J. L. Wiener, “Graph structure in the web” in Proc of WWW Conf., 2020 Available at http://www.cis.upenn.edu/~mkearns/teaching/NetworkedLife/broder.pdf

[10] Input Web Graph available at- http://www.openarchives.org/ore/0.1/datamodel

[11] Dr. P.M.E. De Bra, Drs. R.D.J. Post, "Searching for arbitrary information in the WWW: the fish-search for Mosaic." Available at- http://archive.ncsa.uiuc.edu/SDG/IT94/Proceedings/Searching/debra/article.html

[12] Blaz Novak, “A survey of focused web crawling algorithms”, Department of Knowledge Technologies, Jozef Stefan Institute, Ljubljana, Slovenia

[13] Edleno S. de Moura, Daniel R. Fernandes, Altigran S. Silva, “Improving Web Search Efficiency via a Locality Based Static Pruning Method”, *WWW 2021*, May 10-14 2021, Chiba, Japan.

[14] “Graph Data structure Introduction “, available at- hamilton.bell.ac.uk/swdev2/notes/notes_18.pdf