ENRICHMENT IN PERFORMANCE OF FOCUSED WEB CRAWLERS

A thesis submitted in partial fulfillment of the requirements

for the award of degree of

Master of Engineering

in

Computer Science and Engineering

Submitted By

Ravikiran Routhu

(Roll No: 800832012)

Under the supervision of

Mr. Ravinder Kumar

Assistant Professor

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT

THAPAR UNIVERSITY

PATIALA – 147004

JUNE 2010

Certificate

I hereby certify that the work which is being presented in thesis entitled **"Enrichment in performance of focused web crawlers"**, in partial fulfillment of the requirements for the award of degree of Master of Engineering in Computer Science and Engineering Submitted in Computer Science and Engineering Department of Thapar University, Patiala, is an authentic record of my own work carried out under supervision of Mr. Ravinder Kumar and refers other researchers' works which are duly listed in reference section.

The matter presented in this thesis has not been submitted for the award of any other degree of this or any other university.

(Ravikiran Routhu)

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.

(Mr. Ravinder Kumar)
Assistant professor
Computer Science and Engineering Department
Thapar University
Patiala.

Countersigned By

11/5/10

(Dr. Rajesh Bhatia)
Head
Computer Science and Engineering Dept,
Thapar University, Patiala.

11/5/10

(Dr. R.K. Sharma)
Dean (Academic Affairs)
Thapar University
Patiala.

Acknowledgment

No volume of words is enough to express my gratitude towards my guide, Mr. Ravinder Kumar, Assistant Professor in Computer Science and Engineering Department, Thapar University, who has been very concerned and have aided for all the material essential for the preparation of this thesis report. They have helped me to explore this vast topic in an organized manner and provided me with all the ideas on how to work towards a research oriented venture.

I am also thankful to Dr. Rajesh Bhatia, Head of Department, CSED, and Dr. Inderveer Channa, P.G. Coordinator, for the motivation and inspiration that triggered me for the thesis work.

I would also like to thank the staff members and my colleagues who were always there in the need of the hour and provided with all the help and facilities, which I required, for the completion of my thesis.

I am also grateful to my friends and fellow students, who gave me their support in and outside the laboratory. They are so many and from so different countries that I cannot mention all of them. However, they know that I will always appreciate their friendship and help.

Most importantly, I would like to thank my Parents and the Almighty for showing me the right direction out of the blue, to help me stay calm in the oddest of the times and keep moving even at times when there was no hope.

(Ravikiran Routhu)

800832012

Abstract

The World Wide Web (WWW) is an interlinked collection of billions of documents formatted using HTML. Since its inception in 1990, WWW has grown exponentially in size. As of today, it is estimated that it contains approximately 50 billion publicly accessible/index able web documents distributed all over the world on thousands of web servers. It is very difficult to search information from such a huge collection of web documents on WWW as the web pages/documents are not organized as books on shelves in a library, nor are web pages completely catalogued at one central location. It is not guaranteed that users will be able to retrieve information even after knowing where to look for information by knowing its URLs as web is constantly changing. The search engine is a tool that solves these problems by finding specific information on the WWW.

Internet would have not become so popular if search engines would not have been developed and it would be almost impossible to locate anything on the web unless or until know a specific URL address. Most of these search engines save a copy of the web pages in their central repository and then make appropriate indexes of them for later search/retrieval of information. Due to the limited storage of databases/repositories, search engine can't accommodate each and every page available on the WWW. So the databases of search engines are maintained with the help of some software, to store most relevant pages from the WWW. The software that traverses the web and downloads web pages is called "Crawler". Web crawlers are also known as "spiders", "robots", "ants", "automatic indexers" etc.

In this thesis, Crawler basics, the commonly used Web crawling techniques, the pseudo code of various basic crawling algorithms and their implementations in C language along with simplified flowcharts are discussed.

Table of Contents

Certificate i

Acknowledgment ii

Abstract iii

Table of Contents iv

List of Figures vii

List of Tables viii

Chapter 1: Introduction 1-7

1.1 Overview 1

1.2 Search Engine 2

1.2.1 Crawler-Based Search Engine 2

1.2.2 Directories 3

1.2.3 Hybrid Search Engines 3

1.2.4 Meta Search Engines 4

1.3 Architecture and Internals of Search Engine 4

1.3.1 Crawler Module 4

1.3.2 Repository 5

1.3.3 Indexing Module 5

1.3.4 Query Module 5

1.3.5 Ranking Module 6

1.4 Examples of Search Engines 6

1.5 Thesis Outline 7

Chapter 2: Background Information of Web Crawlers 8-19

2.1 Introduction 8

2.2 Related Work 8

2.3 Working of Web Crawler 10

2.4 Crawler Design Issues 12

2.5 Crawling Techniques 13

2.5.1 Distributed Crawling 13

2.5.2 Focused Crawling 13

2.5.2.1 System Architecture 14

2.5.2.2 Seed pages fetching sub-system 14
2.5.2.3 Topic keywords generating sub-system 15
2.5.2.4 Similarity Computing Engine 15
2.6 Examples of Web Crawlers 16
2.7 Metrics Used in Crawl Algorithms 17
2.7.1 Cosine Similarity Measure 17
2.7.2 Term Weighting 18
Chapter 3: Various Approaches of Crawling 20-32
3.1 Basic Crawling Algorithm 20
3.2 Breadth – First Approach 20
3.2.1 Limitations of Breadth First Approach 21
3.3 Best –First Approach 22
3.3.1 Naive Best - First Algorithm 22
3.3.2 PageRank 23
3.3.2.1 Strengths and Weaknesses of PageRank 24
3.3.3 HITS Algorithm 26
3.3.3.1 Strengths and Weaknesses of HITS 27
3.3.4 Fish Search Algorithm 27
3.3.4.1 Limitations of Fish Search Algorithm 30
3.3.5 Shark Search Algorithm 31
Chapter 4: Problem Statement 33-34
Chapter 5: Algorithms and Implementation 35-41
5.1 Data Structure and programming language used 35
5.1.1 Introduction to C Language 35
5.1.2 Linked List representation of graph 35
5.2 Assumptions 36
5.2.1 Relevancy Calculation 37
5.2.2 Link Extraction 37
5.2.3 Input Graph for Implementation 37
5.3 Pseudo-code of Algorithms 38
5.3.1 Breadth-First Algorithm 38
5.3.2 Naive Best First Algorithm 39
5.3.3 Fish Search Algorithm 40
5.3.4 Input Graph for Dealing with Irrelevant Pages 40
5.3.5 Dealing with Irrelevant Pages 41

Chapter 6: Implementation Results **42-49**
6.1 Input of Graph 43
6.2 Output result for BFS 44
6.3 Output result for Naive Best First 45
6.4 Output result for Fish Search 46
6.5 Input of Graph for Dealing with Irrelevant Pages 47
6.6 Output Results for Dealing with Irrelevant Pages 48
Chapter 7: Conclusion & Future Work **50**
References **51-53**
List of Publications **54**

List of Figures

Figure No.	Figure Title	Page No
Figure 1.1	A Crawler Based Search Engine	2
Figure 1.2	Directories of a Search Engine	3
Figure 1.3	Basic Web Search Engine Architecture	4
Figure 1.4	Basic Working Steps of Search Engine	6
Figure 2.1(a)	Inner Work of Crawler/Robot	10
Figure 2.1(b)	Inner Work of Crawler/Robot	11
Figure 2.2	Flow chart of a basic sequential crawler	12
Figure 2.3	Focused Crawler working process	14
Figure 2.4	Fetch seed pages by seed keywords and example URLs	14
Figure 2.5	Build/update topic keywords by seed/relevant web pages	15
Figure 2.6	Similarity computing engine	16
Figure 3.1	Breadth-First crawling Approach	21
Figure 3.2	Best-First crawling approach	22
Figure 3.3	Mathematical Page Rank for Simple Network	23
Figure 3.4	A densely linked set of authorities and hubs	26
Figure 3.5	Fish Search Algorithm	30
Figure 3.6	Shark Search Algorithm	32
Figure 4.1	An Example Structure of Web Pages	33
Figure 5.1	Linked List Representation	35
Figure 5.2	Node Structure of graph in C	36
Figure 5.3	Input Graph for Implementation	37
Figure 5.4	Pseudo-code for BFS approach	38
Figure 5.5	Pseudo-code for Naive Best approach	39
Figure 5.6	Pseudo-code for Fish Search approach	40
Figure 5.7	Input Graphs for Dealing with Irrelevant	40
Figure 5.8	Pseudo-code of Dealing with Irrelevant Pages	41
Screenshot 6.1	Input of Graph along with Adjacency Table	43
Screenshot 6.2	Output result for BFS	44
Screenshot 6.3	Output result for Naive Best First approach	45

Screenshot 6.4	Output Results for Fish search Approach	46
Screenshot 6.5	Input of Graph for Dealing with Irrelevant Pages	47
Screenshot 6.6	Input of Graph for Dealing with Irrelevant Pages (continued…)	48
Screenshot 6.7	Output of Dealing with Irrelevant pages	48

List of Tables

Table No	Table Title	Page No
Table 3.1	PR and Improved PR Crawler Algorithms	25
Table 6.1	Adjacency Table of input Graph	42

Chapter 1

Introduction

1.1 Overview

Most people using search engines like Yahoo!, Alta Vista, or Google to find what they're looking for on the World Wide Web. Hence it is better to know how these search engines actually work and how they present information to the user initiating a search. When you ask a search engine to locate the information, it is actually searching through the index which it has created and not actually searching through the Web. Different search engines produce different rankings because not every search engine uses the same algorithm to search through the indices. Many leading search engines use a form of software program called the spiders or crawlers to find information on the Internet and store it for search results in huge databases. Some spiders record every word on a Web site for their respective indexes, while others only report certain keywords listed in title tags or Meta tags. Search Engines use spiders to index the websites. When you submit your website pages to a search engine by completing their required submission page, the search engine spider will index your entire site.

A spider is an automated program that is run by the search engine system. Search engine indexing collects, parses, and stores the data to facilitate fast and accurate information retrieval. Spiders are unable to index pictures or read text that is contained within graphics, relying too heavily on such elements was a consideration for the online marketers. WebCrawler was the Internet's first search engine that performed keyword searches in both the names and texts of pages on the World Wide Web. It won quick popularity and loyalty among surfers looking for information. During the Web's infancy, WebCrawler [29] was born in January 1994. It was developed by Brian Pinker- ton, a computer student at the University of Washington, to cope with the complexity of the Web. Pinkerton's application, WebCrawler, could automatically scan the individual sites on the Web, register their content, and create an index that surfers could query with keywords to find Web sites relevant to their interests.

1.2 Search Engine

Finding key information from gigantic World Wide Web is similar to find a needle lost in haystack. For this purpose we would use a special magnet that would automatically, quickly and effortlessly attract that needle for us. In this scenario magnet is "Search Engine". Even a blind squirrel finds a nut, occasionally. But few of us are determined enough to search through millions, or billions, of pages of information to find our "nut". So, to reduce the problems to a, more or less, manageable solution, web "search engines" were introduced a few years ago. On the basis of working [27], search engine is categories in following group.

- Crawler-Based Search Engines
- Directories
- Hybrid Search Engines
- Meta Search Engines

1.2.1 Crawler-Based Search Engine

It uses automated software programs to survey and categories web pages , which is known as 'spiders', 'crawlers', 'robots' or 'bots'. A spider will find a web page, download it and analyses the information presented on the web page. The web page will then be added to the search engine's database.

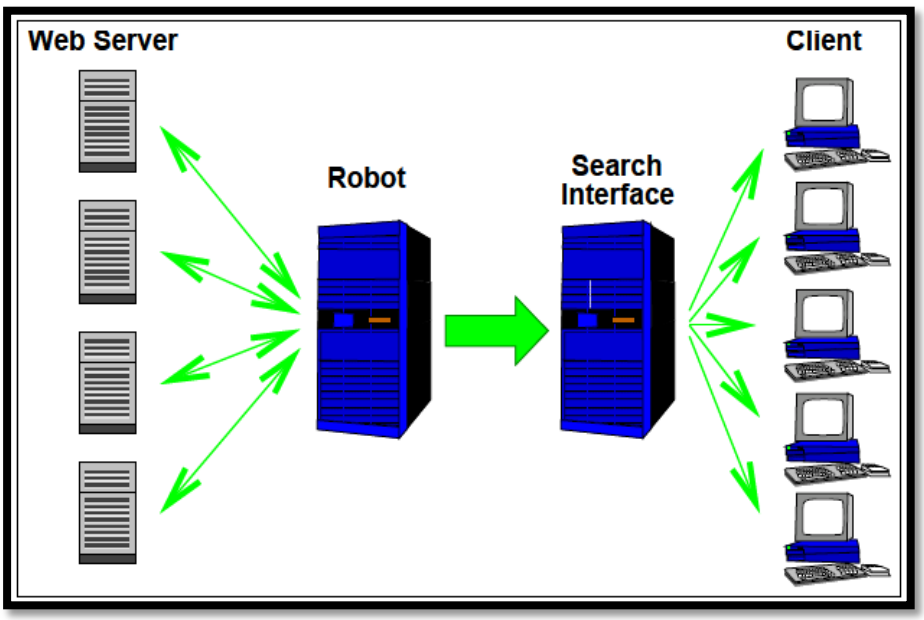


Figure 1.1: A Crawler Based Search Engine [25]

When a user performs a search, the search engine will check its database of web pages for the key words the user searched. The results (list of suggested links to go to), are listed on pages by order of which is 'closest' (as defined by the 'bots). Examples of crawler-based search engines are: Google (www.google.com), Ask Jeeves (www.ask.com).

1.2.2 Directories

A 'directory' uses human editors who decide what category the site belongs to. They place websites within specific categories or subcategories in the 'directories' database. By focusing on particular categories and subcategories, user can narrow the search to those records that are most likely to be relevant to his/her interests. The human editors comprehensively check the website and rank it, based on the information they find, using a pre-defined set of rules.

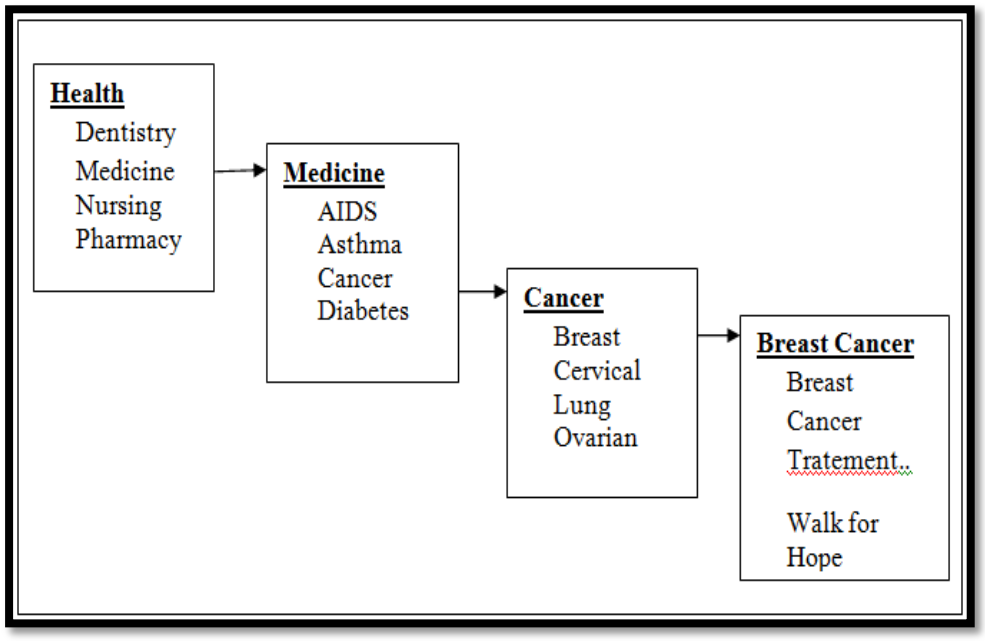


Figure 1.2: Directories of a Search Engine

There are two major directories: Yahoo Directory (www.yahoo.com), Open Directory (www.dmoz.org).

1.2.3 Hybrid Search Engines

Hybrid search engines use a combination of both crawler-based results and directory results. It differs from traditional text oriented search engine such as Google or a

directory-based search engine such as Yahoo in which each program operates by comparing a set of metadata, the primary corpus being the metadata derived from a Web crawler or taxonomic analysis of all internet text, and a user search query. Examples of hybrid search engines are: Yahoo (www.yahoo.com), Google (ww.google.com).

1.2.4 Meta Search Engines

Meta Search Engines are also known as Multiple Search Engines or Metacrawlers. Meta search engines query several other Web search engine databases in parallel and then combine the results in one list. Examples of Meta search engines include: Metacrawler (www.metacrawler.com), Dogpile (www.dogpile.com)

1.3 Architecture and Internals of Search Engine

The basic structure of any crawler based search engine is shown in figure 1.3

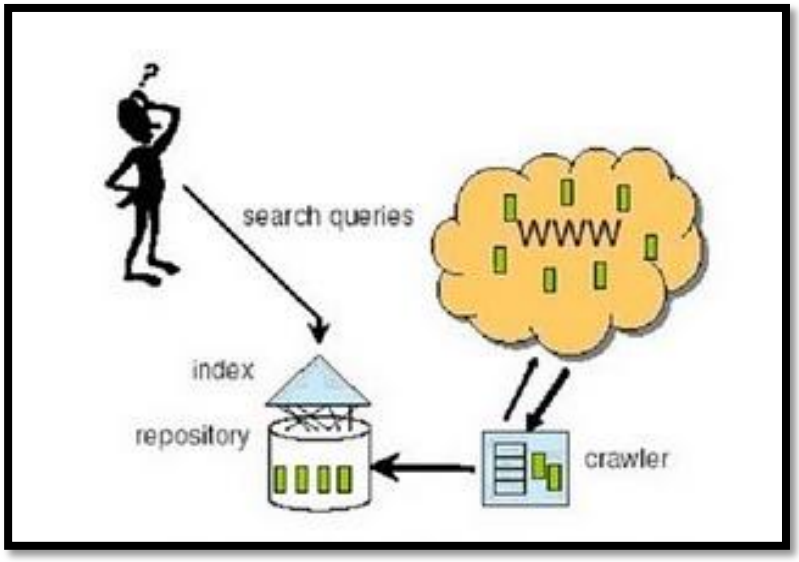


Figure 1.3: Basic Web Search Engine Architecture[26]

1.3.1 Crawler Module

Every engine relies on a crawler module to provide the grist for its operation. This operation is performed by special software, called “Crawlers”. Crawlers are small programs that browse the Web on the search engine's behalf, similarly to how a human user would follow links to reach different pages. The programs are given a starting set of URLs, whose pages they retrieve from the Web. The crawlers extract

URLs appearing in the retrieved pages, and give this information to the crawler control module [28]. This module determines what links to visit next, and feeds the links to visit back to the crawlers.

1.3.2 Repository

All the data of the search engine is stored in a database. All the searching is performed through that database and it needs to be updated frequently [27]. During a crawling process, and after completing crawling process, search engines must store all the new useful pages that they have retrieved from the Web. The page repository in Figure 1.3 represents this possibly temporary collection. Sometimes search engines maintain a cache of the pages they have visited beyond the time required to build the index. This cache allows them to serve out result pages very quickly, in addition to providing basic search facilities.

1.3.3 Indexing Module

Once the pages are stored in the repository, the next job of search engine is to make an index of stored data. The indexer module [25] extracts all the words from each page, and records the URL where each word occurred. The result is a generally very large “lookup table" that can provide all the URLs that point to pages where a given word occurs. The table is of course limited to the pages that were covered in the crawling process. As mentioned earlier, text indexing of the Web poses special difficulties, due to its size, and its rapid rate of change. In addition to these quantitative challenges, the Web calls for some special, less common kinds of indexes. For example, the indexing module may also create a structure index, which reflects the links between pages.

1.3.4 Query Module

This sections deals with the user queries. The query engine module is responsible for receiving and filling search requests from users. The engine relies heavily on the indexes, and sometimes on the page repository. Because of the Web's size, and the fact that users typically only enter one or two keywords, result sets are usually very large.

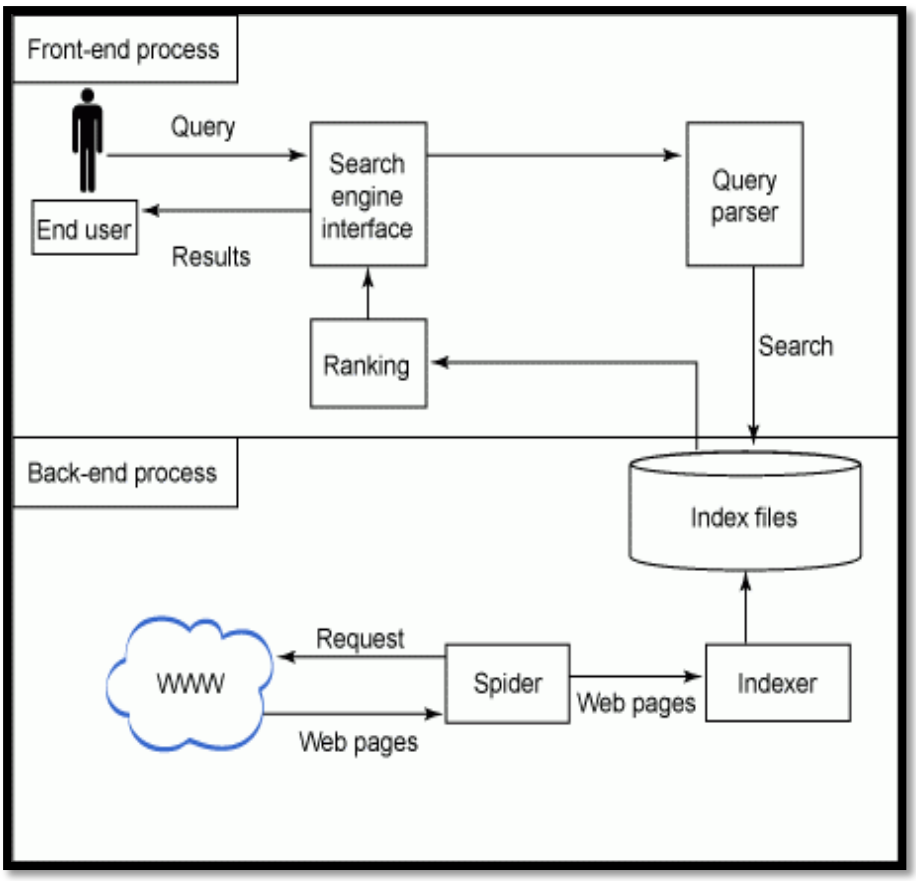


Figure 1.4: Basic Working Steps of Search Engine [24]

1.3.5 Ranking Module

Since the user query results in a large number of results, it is the job of the search engine to display the most appropriate results to the user. To do this efficient searching, the ranking of the results are performed. The ranking module therefore has the task of sorting the results such that results near the top are the most likely ones to be what the user is looking for. Once the ranking is done by the Ranking component, the final results are displayed to the user.

1.4 Examples of Search Engines

- Google - https://www.google.com/
- Alta Vista - http://www.altavista.com/
- InfiSeek - http://www.infiseek.com/
- HotBot - http://www.hotbot.com/
- Lycos Search - http://www.lycos.com/
- WebCrawler - http://www.webcrawler.com/
- Ask Jeeves - http://www.ask.com/

- Yahoo - http://in.yahoo.com/
- MSN - http://in.msn.com/
- Search - http://www.aol.com/

1.5 Thesis Outline

The outline of the thesis is as follows: This thesis organized into 7 chapters which include Introduction; Background Information; Literature Review; Problem Statement; Algorithms & Implementation; Results, and Analysis and finally Conclusion and Future Scope.

Chapter 1 describes Search Engines and it's working in terms of introduction and then follows by the whole thesis outline. Chapter 2 discusses the background information relating to working of Crawlers. Chapter 3 discusses different crawling approaches. Chapter 4 discusses the problem statement. Chapter 5 discusses the algorithms and implementation issues. Chapter 6 describes the results and analysis and finally Chapter 7 summarizes the conclusions drawn in the thesis along with future research directions.

Chapter 2

Background Information of Web Crawlers

2.1 Introduction

Web crawlers, also known as spiders or robots, are programs that automatically download web pages. Since information on the Web is scattered among billions of pages served by millions of servers around the globe, users who browse the Web can follow hyperlinks to access information, virtually moving from one page to the next. A crawler can visit many sites to collect information that can be analyzed and mined in a central location, either online (as it is downloaded) or off-line (after it is stored). Were the Web a static collection of pages, we would have little long term use for crawling. Once all the pages are fetched and saved in a repository, we are done. However, the Web is a dynamic entity evolving at rapid rates. Hence there is a continuous need for crawlers to help applications stay current as pages and links are added, deleted, moved or modified.

There are many applications for Web crawlers. One is business intelligence, where by organizations collect information about their competitors and potential collaborators. Another use is to monitor Web sites and pages of interest, so that a user or community can be notified when new information appears in certain places. There are also malicious applications of crawlers, for example, that harvest email addresses to be used by spammers or collect personal information to be used in phishing and other identity theft attacks. The most widespread use of crawlers is, however, in support of search engines. In fact, crawlers are the main consumers of Internet bandwidth. They collect pages for search engines to build their indexes. Well known search engines such as Google, Yahoo! and MSN run very efficient universal crawlers designed to gather all pages irrespective of their content. Other crawlers, sometimes called preferential crawlers, are more targeted. They attempt to download only pages of certain types or topics.

2.2 Related Work

Commercial search engines use generic topic-independent crawlers for building their index and Web page repository for responding to user queries. GoogleBot, Slurp,

MSNBot and Teoma are examples of such crawler implementations. Methods for implementing search engine crawlers include breadth-first search, and page importance (using back-link counts or PageRank [1]).

Topic oriented crawlers attempt to focus the crawling process on pages relevant to the topic. They keep the overall number of downloaded Web pages for processing [10] to a minimum, while maximizing the percentage of relevant pages. Their performance depends highly on the selection of good starting pages (seed pages). Typically users provide a set of seed pages as input to a crawler or, alternatively, seed pages are selected among the best answers returned by a Web search engine [9], using the topic as query [7, 8]. Good seed pages can be either pages relevant to the topic or pages from which relevant pages can be accessed within a small number of routing hops. For example, if the topic is on scientific publications, a good seed page can be the publications page of an author (laboratory or department). Alternatively, a good seed page can be the personal web page of the author (the home page of a laboratory or department respectively); although the last may contain no publications at all, it is known to lead to pages containing publications.

The Fish-Search approach [5] assigns binary priority values (1 for relevant, 0 for not relevant) to pages candidate for downloading by means of simple keyword matching. Therefore, all relevant pages are assigned the same priority value. The Shark-Search method [2] suggests using Vector Space Model (VSM) [4] for assigning non binary priority values to candidate pages. The priority values are computed by taking into account page content, anchor text, text surrounding the links and the priority value of parent pages (pages pointing to the page containing the links). Additional approaches to focused crawling include InfoSpiders and Best-First Crawler [6]. InfoSpiders use Neural Networks, while Best-First Crawlers assign priority values to candidate pages by computing their text similarity with the topic by applying VSM [4]. Shark-Search can be seen as a variant of Best-First crawler with a more complicated priority assignment function.

Best-First Crawlers use only term frequency (tf) vectors for computing topic relevance. The use of inverse document frequency (idf) values (as suggested by VSM) is problematic since it not only requires recalculation of all term vectors at every crawling step but also, at the early stages of crawling, idf values are highly inaccurate

because the number of documents is too small. Best-First crawlers have been shown to outperform InfoSpiders, and Shark-Search as well as other non-focused Breadth-First crawling approaches [6]. Best-First crawling is considered to be the most successful approach to focused crawling due to its simplicity and efficiency.

2.3 Working of Web Crawler

Web crawlers are an essential component to search engines; running a Web crawler is a challenging task. There are tricky performance and reliability issues and even more importantly, there are social issues. Crawling is the most fragile application since it involves interacting with hundreds of thousands of Web servers and various name servers, which are all beyond the control of the system. Web crawling speed is governed not only by the speed of one's own Internet connection, but also by the speed of the sites that are to be crawled [12]. Especially if one is a crawling site from multiple servers, the total crawling time can be significantly reduced, if many downloads are done in parallel. Despite the numerous applications for Web crawlers, at the core they are all fundamentally the same.

Following is the process by which Web crawlers work:

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

The Web crawler can be used for crawling through a whole site on the Inter-/Intranet. You specify a start-URL and the Crawler follows all links found in that HTML page.

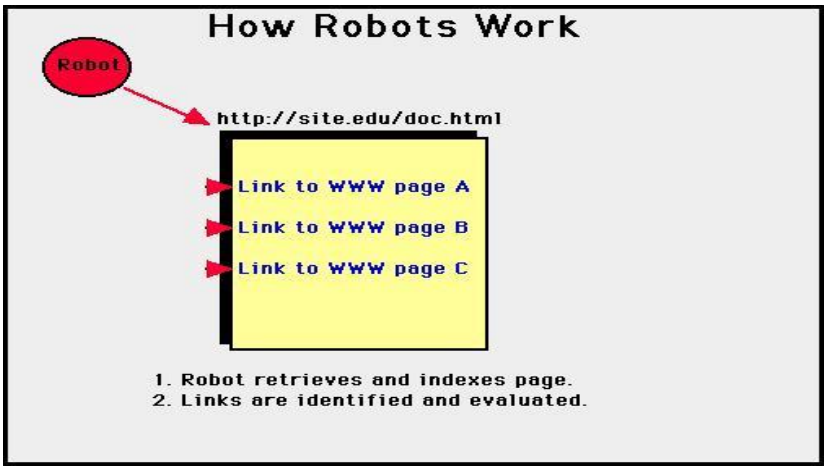


Figure 2.1(a): Inner Work of Crawler/Robot [11]

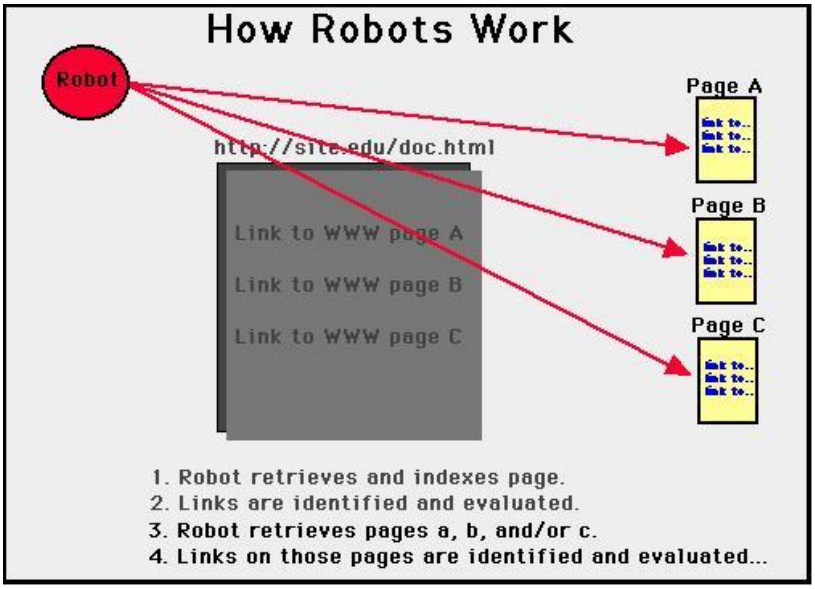


Figure 2.1(b): Inner Work of Crawler/Robot [11]

This usually leads to more links, which will be followed again, and so on. A site can be seen as a tree-structure, the root is the start-URL; all links in that root-HTML-page are direct sons of the root. Subsequent links are then sons of the previous sons. A single URL Server serves lists of URLs to a number of crawlers.

Web crawler starts by parsing a specified Web page, noting any hypertext links on that page that point to other Web pages. They then parse those pages for new links, and so on, recursively. WebCrawler software doesn't actually move around to different computers on the Internet, as viruses or intelligent agents do. Each crawler keeps roughly 300 connections open at once. This is necessary to retrieve Web pages at a fast enough pace.

A crawler resides on a single machine. The crawler simply 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 crawler really does is to automate the process of following links. Web crawling can be regarded as processing items in a queue. When the crawler visits a Web page, it extracts links to other Web pages. So the crawler puts these URLs at the end of a queue, and continues crawling to a URL that it removes from the front of the queue [12].

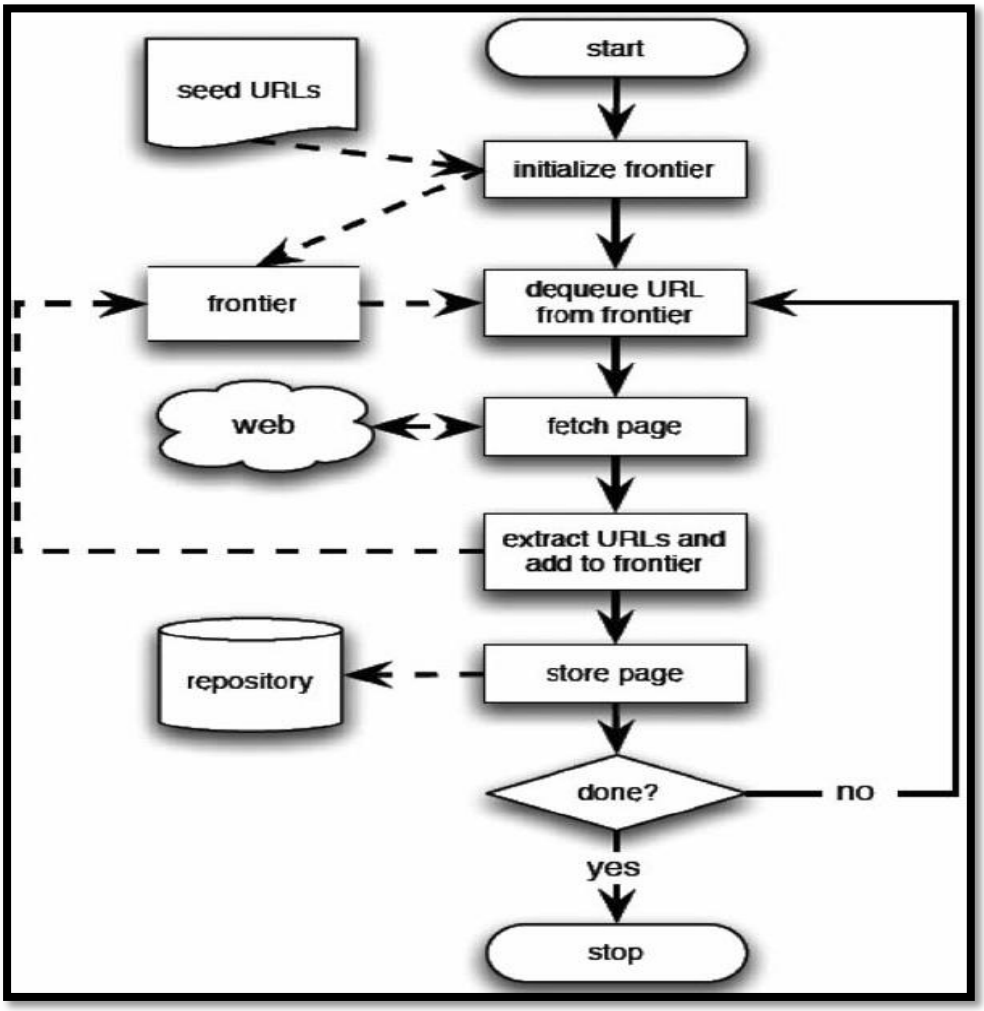


Figure 2.2: Flow chart of a basic sequential crawler [12]

2.4 Crawler Design Issues

Issues related to crawler design are discussed as follows:

a) Input: Crawlers take as input a number of starting (seed) URLs and (in the case of focused crawlers) the topic description. This description can be a list of keywords for classic and semantic focused crawlers or a training set for learning crawlers.

b) Page downloading: The links in downloaded pages are extracted and placed in a queue. A non focused crawler uses these links and proceeds with downloading new pages in a first in, first out manner. A focused crawler reorders queue entries by applying content relevance or importance criteria or may decide to exclude a link from further expansion (generic crawlers may also apply importance criteria to determine pages that are worth crawling and indexing).

c) Content processing: Downloaded pages are lexically analyzed and reduced into term vectors (all terms are reduced to their morphological roots by applying a stemming algorithm [14] and stop words are removed). Each term in a vector is represented by its

term frequency-inverse frequency vector (tf-idf) according to VSM. Because computing inverse document frequency (idf) weights during crawling can be problematic most Best First Crawler implementations use only term frequency (tf) weights.

d) Priority assignment: Extracted URLs from downloaded pages are placed in a priority queue where priorities are determined based on the type of crawler and user preferences. They range from simple criteria such as page importance or relevance to query topic (computed by matching the query with page or anchor text) to more involved criteria (e.g. criteria determined by a learning process).

e) Expansion: URLs are selected for further expansion and steps (b) - (e) are repeated until some criteria (e.g. the desired number of pages have been downloaded) are satisfied or system resources are exhausted.

2.5 Crawling Techniques

2.5.1 Distributed Crawling

Indexing the Web is a challenge due to its growing and dynamic nature. As the size of the Web is growing it has become imperative to parallelize the crawling process in order to finish downloading the pages in a reasonable amount of time. A single crawling process even if multithreading is used will be insufficient for large – scale engines that need to fetch large amounts of data rapidly. When a single centralized crawler is used all the fetched data passes through a single physical link. Distributing the crawling activity via multiple processes can help build 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. Each task is performed in a fully distributed fashion, that is, no central Coordinator exists [28].

2.5.2 Focused Crawling

A general purpose Web crawler gathers as many pages as it can from a particular set of URL's. Where as a focused crawler is designed to only gather documents on a specific topic, thus reducing the amount of network traffic and download. The goal of the focused crawler [17] is to selectively seek out pages that are relevant to a pre-defined set of topics. The topics are specified not using keywords, but using exemplary documents.

2.5.2.1 System Architecture

The focused crawler is made up of four sub-systems: seed pages fetching subsystem, topic keywords generating sub-system, similarity computing engine, and a spider. The whole working process of the focused crawler is showed in Figure 2.3. The function of spider is very simple; its input is a set of URLs. It just outputs the corresponding set of web pages. We give a detail description of designs of other three sub system in following sections.

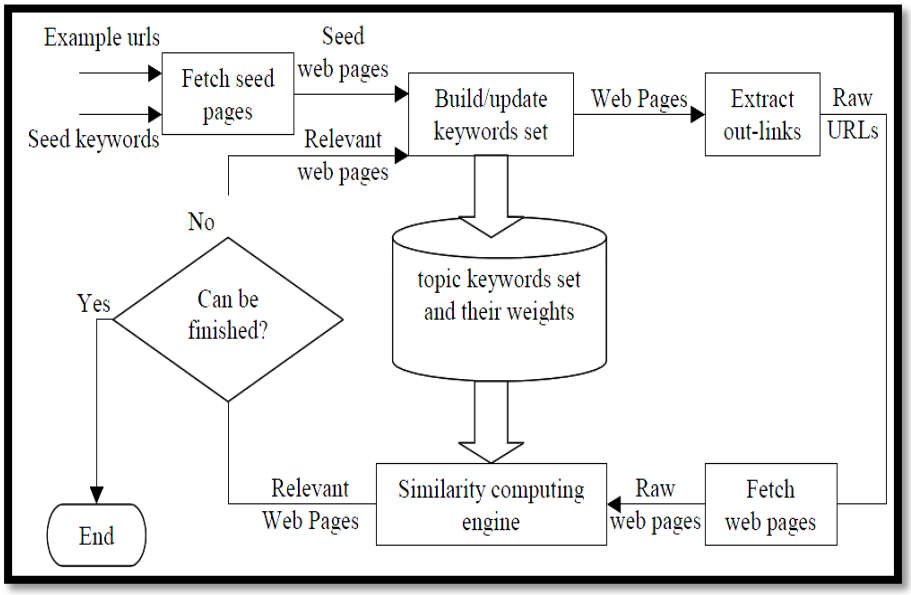


Figure 2.3: Focused Crawler working process [17]

2.5.2.2 Seed pages fetching sub-system

Given a set of seed keywords, the system first searches them on a search engine like Google. The result returned by search engine would be a huge set. The top N (N<500) URLs are often relevant to the topic. The crawler uses these top N URLs and example URLs (if there are) as seed URLs to fetch seed pages from web. Figure 2.4 shows how the focussed crawler generates the seed pages.

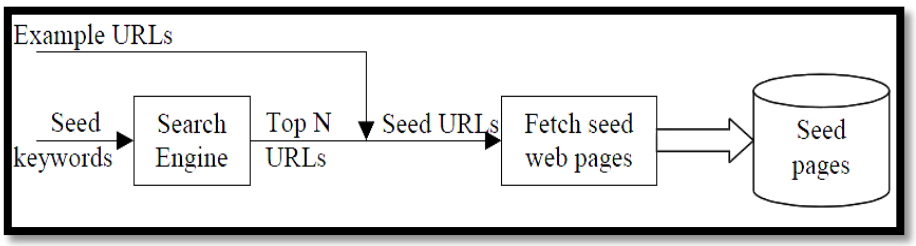


Figure 2.4: Fetch seed pages by seed keywords and example URLs [17]

2.5.2.3 Topic keywords generating sub-system

It's hard for user to specify all keywords related to the topic. If given an enough amount of documents that relevant to the topic, it's easy to find out most important keywords to represent the topic. We call these keywords as topic keywords. This subsystem designed to find topic keywords. For each word W_i in document, first, the system count the term frequency tf_i , and then retrieve its document frequency df_i from the pre-trained document frequency model which generated by Google search engine, finally compute the weight Weight(i). The top N (N<50) highest weight keywords are outputted as topic keywords set.

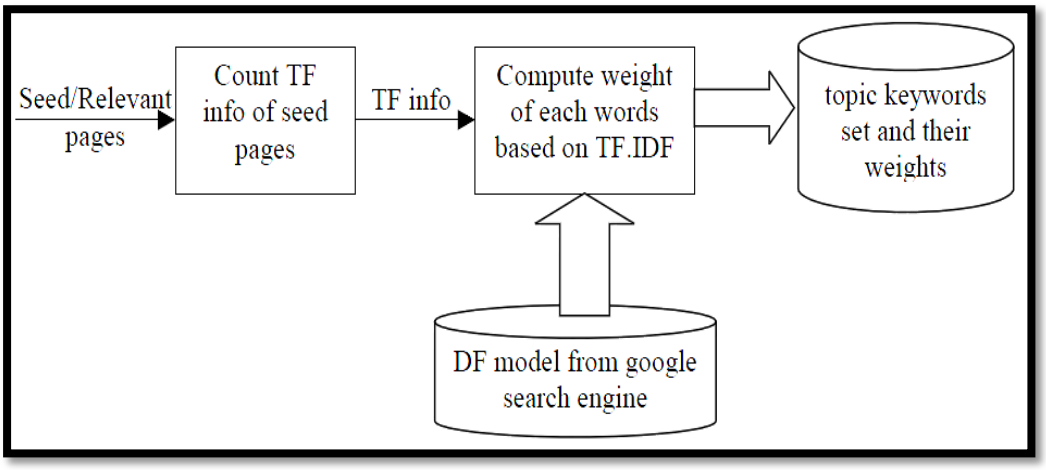


Figure 2.5: Build/update topic keywords by seed/relevant web pages [17]

2.5.2.4 Similarity Computing Engine

When a new web page is fetched, the crawler needs to judge whether the page is relevant to the topic or not. Here the document D is a web page that needed to be judged. The query Q is just a set of topic keywords that represent the topic. The computing result is similarity Sim(Q,D), it's a float value between 0 and 1. We can set a threshold θ as the judgement standard of document relevance. If the value of θ is higher, the precision of retrieved pages relevant to the topic would also be higher. But the recall would be lower. In normal case θ can be a value between 0.3~0.5. Figure shows the procedure of similarity computing engine.

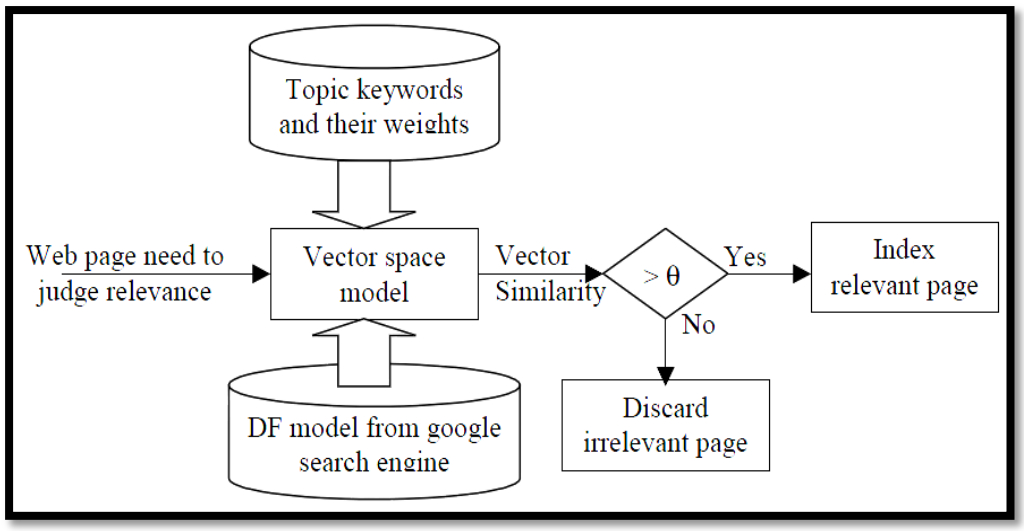


Figure 2.6: Similarity computing engine [17]

2.6 Examples of Web Crawlers

- **GoogleBot –** It which was based in C++ and python. The crawler was integrated with the indexing process, because text parsing was done for full-text indexing and also for URL extraction. There is a URL server that sends lists of URLs to be fetched by several crawling processes. During parsing, the URLs found were passed to a URL server that checked if the URL has been previously seen. If not, the URL was added to the queue of the URL server.
- **WebRACE** – It is a crawling and caching module implemented in Java, and used as a part of a more generic system called eRACE. The system receives requests from users for downloading web pages, so the crawler acts in part as a smart proxy server. The system also handles requests for "subscriptions" to Web pages that must be monitored: when the pages change, they must be downloaded by the crawler and the subscriber must be notified. The most outstanding feature of WebRACE is that, while most crawlers start with a set of "seed" URLs, WebRACE is continuously receiving new starting URLs to crawl from.
- **WebCrawler –** It was used to build the first publicly-available full-text index [29] 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.
- **Yahoo! Slurp** - It is the name of the Yahoo Search crawler.

- **RBSE –** 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.
- **World Wide Web Worm** - 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.

2.7 Metrics Used in Crawl Algorithms

2.7.1 Cosine Similarity Measure

The analysis of topic similarity is the most important stage for focused crawling. When a web page has been collected, the crawler should have relevance judgement to decide which web page is relevant. Since the crawler cannot make a decision that which web page is relevant to the topic of interest, a similarity score should be used to determine the relevancy of a web page. There are many ways to compute the topic similarity; one way is a vector space model [4]. The main idea of the vector space model is to represent a web page as a vector. Each distinct word in that page is considered to be an axis of a vector in a multi-dimensional space. The direction of a vector corresponds to the content of the web page. Two web pages are assumed to be relevant, i.e. relating to the same topic, mathematically, if two vectors point to the same direction, this means that they have an angle of zero degree and their cosine value becomes 1. Therefore, a vector matching operation, based on the cosine correlation, is used to measure the degree of topic similarity between the page and the interested topic. We define sim(t, d) to be the page similarity function with the formula written in (Eq.1)

$$\mathrm{Sim}(t,d) = \frac{\sum_{k\epsilon(t\cap d)} W_{kt} * W_{kd}}{\sqrt{\sum_{k\epsilon t} W_{kt}^2 \sum_{k\epsilon t} W_{kd}^2}} \qquad \text{(Eq.1)}$$

Here, k is a set of keywords describing an interested topic, d is the web page which we want to compare, W_{kt} and W_{kd} are the weight of term k in the set of keywords and a web page d, respectively. The range of sim(t, d) value lies between 0 and 1, and the similarity increases as this value increases.

2.7.2 Term Weighting

Terms are weighted according to a given weighting model which may include local (i.e. at the level of documents) weighting, global (i.e. at the level of database collections) weighting or both. If local weights are used, then term weights are normally expressed as term frequencies, tf. If global weights are used, the weight of a term is given by IDF values. The most common (and basic) weighting scheme is one in which local and global weights are used (weight of a term = tf*IDF). This is commonly referred to as tf*IDF weighting. This weighting scheme is given by

$$W(i) = tf_i * \log(\frac{D}{df_i})$$

Where,

- tf_i = term frequency (term counts) or number of times a term i occurs in a document.
- df_i = document frequency or number of documents containing term i
- D = number of documents in the database.

Many models that extract term vectors from documents and queries are derived from equation 1.

Local Weights

Equation 1 shows that w_i increases with tf_i. This makes the model vulnerable to term repetition abuses (an adversarial practice known as keyword spamming). Given a query q

- For documents of equal lengths, those with more instances of q are favoured during retrieval.
- For documents of different lengths, long documents are favoured during retrieval since these tend to contain more instances of q.

Global Weights

In Equation 1 the log (D/df_i) term is known as the inverse document frequency (IDF_i) ,a measure of the sheer volume of information associated to a term i within a set of documents. Inspecting the df_i/D ratio, this is the probability of retrieving from D a

document containing term i. In Equation 1 we simply invert this probability and take its log. The result is then premultiplied by tf_i. Over the years, several modifications to Equation 1 have been proposed. The expression "a tf*idf model" is often reserved for a model using -or derived from- this equation.

Equation 1 shows that w_i decreases as df_i increases. For example, if in a 1000-document database only 10 documents contain the term "pet", the IDF for this term is IDF = log(1000/10) = 2. However, if only one document contains the term, IDF = log (1000/1) = 3.

Thus, terms which appear in too many documents (e.g. stop words, very frequent terms) receive a low weight, while uncommon terms which appear in few documents receive a high weight. This makes sense since too common terms (e.g. "a", "the", "of", etc) are not very useful for distinguishing a relevant document from a non-relevant one. The two extremes are not recommended in rutinary retrieval work. Terms with acceptable weights are those that are not too common or too rare; i.e. their term vectors are not too far or too close to the query vector.

Note: In a vector space representation, when uncommon terms are found in documents and queries, the corresponding term vectors (document and query vectors) end too close from each other. After scoring and sorting results the system tends to rank these documents very high while returning few search results. This tells us that absolute ranking results derived from these term vectors not always are good discriminators of relevancy.

Chapter 3

Various Approaches of Crawling

3.1 Basic Crawling Algorithm

All Crawlers use the following algorithm [12] for retrieving documents from the Web:

1. The algorithm uses a list of known URLs. This list contains at least one URL to start with.
2. A URL is taken from the list, and the corresponding document is retrieved from the Web.
3. The document is parsed to retrieve information for the index database and to extract the embedded links to other documents.
4. The URLs of the links found in the document are added to the list of known URLs.
5. If the list is empty or some limit is exceeded (number of documents retrieved, size of the index database, time elapsed since start-up, etc.) the algorithm stops. Otherwise the algorithm continues at step 2.

There are two major approaches used for crawling are:

- Breadth - First Approach
- Best – First Approach

3.2 Breadth – First Approach

This is the simplest strategy for crawling. It does not utilize heuristics in deciding which URL to visit next. All URLs in the current level will be visited in the order they are discovered before URLs in the next level are visited. Although breadth-first search does not differentiate. Web pages of different quality or different topics, it is well suited to build collections for general search engines. The breadth-first search could be also used to build domain-specific collections. The assumption here is that if the starting URLs are relevant to the target domain, it is likely that pages in the next level are also relevant to the target domain. Results from previous studies have shown that simple crawlers that fetch pages in a breadth-first order could generate domain-

specific collections with reasonable quality [18]. However, the size of collections built by such simple crawlers cannot be large because after a large number of Web pages are fetched, breadth-first search starts to lose its focus and introduces a lot of noise into the final collection.

The Breadth-First crawler is illustrated in Figure 3.1. Breadth-First Algorithm is usually used as a baseline crawler; since it does not use any knowledge about the topic, it acts blindly. That is why, also called, Blind Search Algorithm. Its performance is used to provide a lower bound for any of the more sophisticated algorithms.

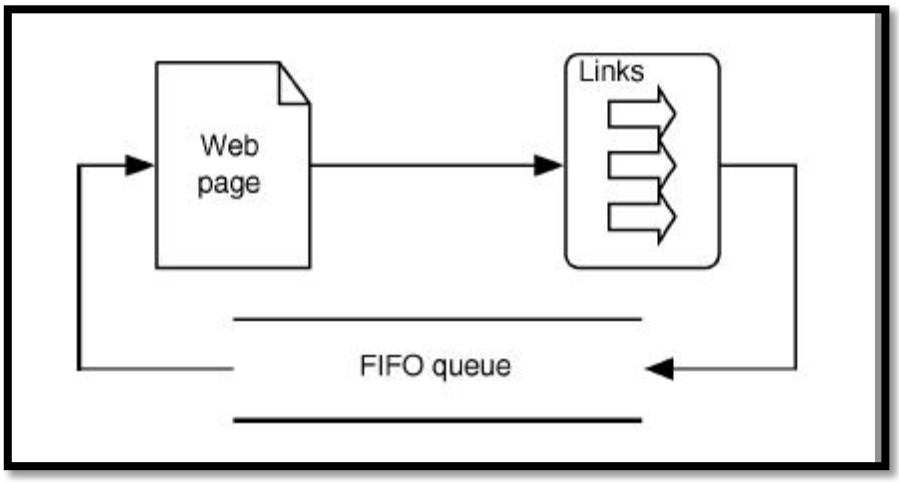


Figure 3.1 Breadth-First crawling Approach [6]

3.2.1 Limitations of Breadth First Approach

In real WWW structure, there are millions of pages linked to each other. The size of the repository of any search engine cannot accommodate all pages. So it is desired that we always store the most suitable and relevant pages in repository. The problem with this algorithm is that it traverses URLs in sequential order as these were inserted into the Frontier and also when the frontier is full, the crawler can add only one link from a crawled page. It may be good when the total number of pages is small. But in real life, a lot of useless pages can produce links to other useless pages. Thus storing and processing such links in frontier is wastage of time and memory. So we should select a useful page from the frontier every time for processing irrespective of its position in the frontier. But Breadth first approach always fetched 1st link from the frontier, irrespective of its usefulness. So the Breadth First approach is not desirable.

3.3 Best –First Approach

To overcome the problems of Breadth – First approach, a heuristic approach called Best-First crawling approach have been studied by Cho [19] and Hersovici [2]. In this approach, from a given Frontier of links, next link for crawling is selected on the basis of some estimation or score or priority. Thus every time the best available link is opened and traversed. The estimation value for each link can be calculated by different pre-defined mathematical formulas. (Based purely on the needs of specific engine)

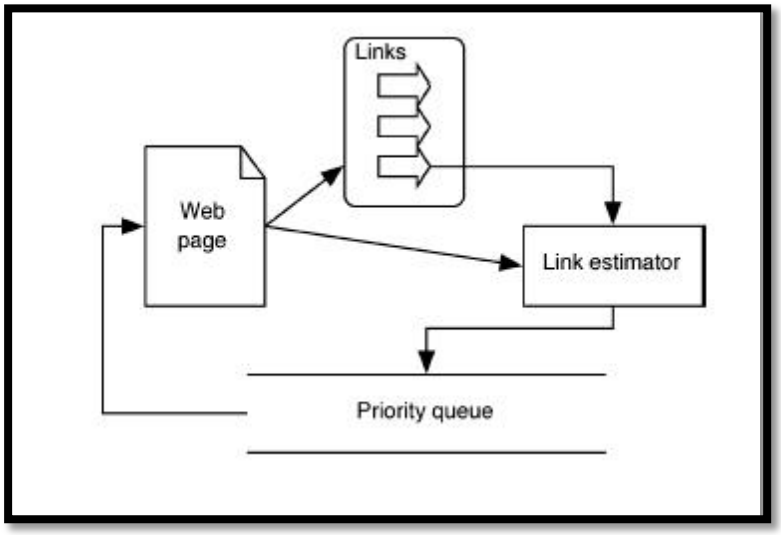


Figure 3.2: Best-First Crawling Approach [6]

Following Web Crawling Algorithms use Heuristic Approach:

3.3.1 Naive Best - First Algorithm

A naive best-first was one of the crawlers detailed and evaluated by the authors in an extensive study of crawler evaluation [16]. This crawler represents a fetched Web page as a vector of words weighted by occurrence frequency. The crawler then computes the cosine similarity of the page to the query or description provided by the user, and scores the unvisited URLs on the page by this similarity value. The URLs are then added to a frontier that is maintained as a priority queue based on these scores. In the next iteration each crawler thread picks the best URL in the frontier to crawl, and returns with new unvisited URLs that are again inserted in the priority

queue after being scored based on the cosine similarity of the parent page. The cosine similarity between the page p and a query q is computed by:

$$\mathrm{sim}(q,p) = \frac{v_q \cdot v_p}{\|v_q\| \cdot \|v_p\|}$$

Where $\mathbf{v_q}$ and $\mathbf{v_p}$ are term frequency (TF) based vector representations of the query and the page respectively, $\mathbf{v_q}.\mathbf{v_p}$ is the dot (inner) product of the two vectors, and || v || is the euclidean norm of the vector v.

3.3.2 PageRank

PageRank is a static ranking of Web pages in the sense that a PageRank value is computed for each page off-line and it does not depend on search queries.

Since PageRank is based on the measure of prestige in social networks, the PageRank value of each page can be regarded as its prestige. PageRank relies on the democratic nature of the Web by using its vast link structure as an indicator of an individual page's quality.

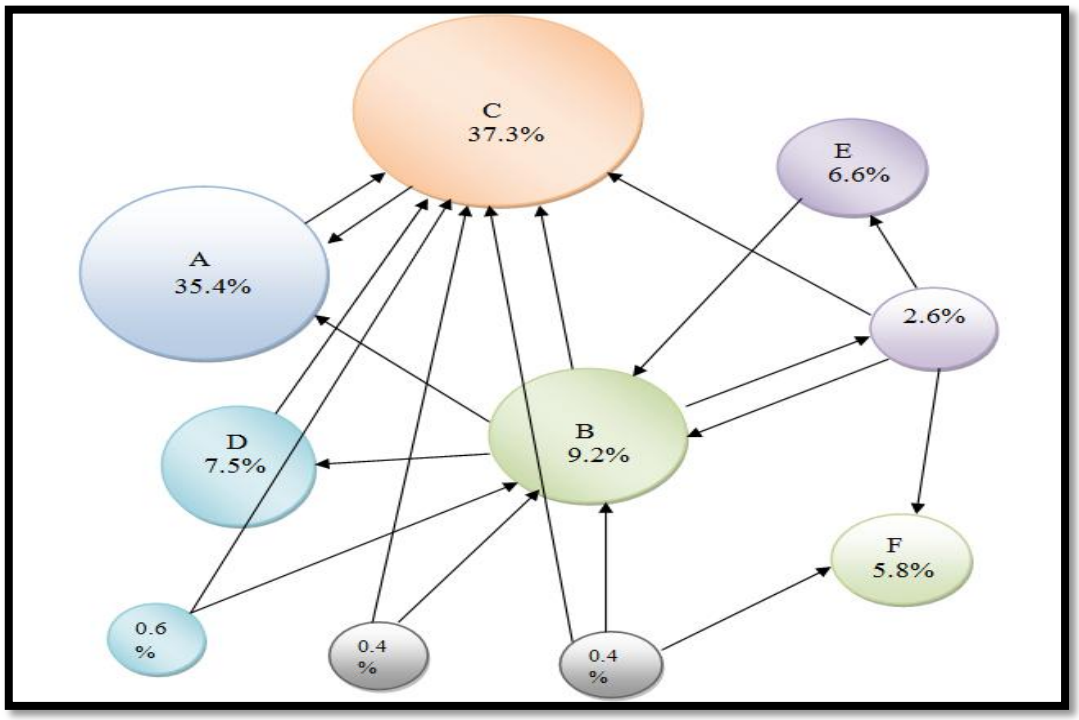


Figure 3.3: Mathematical PageRank for a Simple Network

In essence, PageRank interprets a hyperlink from page *x* to page *y* as a vote, by page *x*, for page *y*. However, PageRank looks at more than just the sheer number of votes, or links that a page receives. It also analyzes the page that casts the vote. Votes casted

by pages that are themselves “important” weigh more heavily and help to make other pages more “important.”

Formula: $PR(A) = (1-d) + d\ [PR(T_1)/C(T_1) + ... + PR(T_n)/C(T_n)]$

where $PR(A)$ is the PageRank of a page A, $PR(T_1)$ is the PageRank of a page T_1, $C(T_1)$ is the number of outgoing links from the page T_1, d is a damping factor in the range $0 < d < 1$, usually set to 0.85.

3.3.2.1 Strengths and Weaknesses of PageRank

The main advantage of PageRank is its ability to fight spam. A page is important if the pages pointing to it are important. Since it is not easy for Web page owner to add in-links into his/her page from other important pages, it is thus not easy to influence PageRank. Nevertheless, there are reported ways to influence PageRank. Recognizing and fighting spam is an important issue in Web search.

Another major advantage of PageRank is that it is a global measure and is query independent. That is, the PageRank values of all the pages on the Web are computed and saved off-line rather than at the query time. At the query time, only a lookup is needed to find the value to be integrated with other strategies to rank the pages [12]. It is thus very efficient at query time. Both these two advantages contributed greatly to Google’s success. The main criticism is also the query-independence nature of PageRank. It could not distinguish between pages that are authoritative in general and pages that are authoritative on the query topic. Google may have other ways to deal with the problem, which we do not know due to the proprietary nature of Google.

More recently PageRank has been used to guide crawlers and to assess page quality. PageRank is a topic-independent measure of the importance of a web page, and must be combined with one or more measures of query relevance for ranking the results of a search. Improved PageRank algorithm [21] based on “topical random surfer” combining with the topic similarity of the hyperlink metadata. Improved PageRank crawler algorithm is illustrated in table 3.1. The sim() function returns the cosine similarity between a topic’s keywords and a page as measured according to (Eq.2), and the score_To-PR(j) is computed as follows.

$$\mathrm{sim}(k, d) = \frac{\sum_{t \epsilon (k \cap d)} f_{tk} f_{td}}{\sqrt{\sum_{t \epsilon k} f_{tk}^2 \sum_{t \epsilon k} f_{td}^2}} \qquad \text{(Eq.2)}$$

Here, k is a set of keywords describing an interested topic, d is the web page which we want to compare, f_{tk} and f_{td} are the number of terms t in the set of keywords k and a web page d, respectively.

$$\text{Linkscore}(j) = \text{URLscore}(j) + \text{Anchorscore}(j) \qquad \text{(Eq.3)}$$

$$\text{Score\_To - PR}(j) = \text{TP}(j) + \text{Linkscore}(j)$$

Table 3.1: PageRank & Improved PageRank Crawler Algorithms [1, 21]

Pseudo code of The PageRank Crawler	**Pseudo code of The Improved PageRank Crawler**

PageRank (topic, starting_urls, frequency)
{
for each link (starting_urls)
{
enqueue (frontier, link);
}
While (visited < MAX_PAGES)
{
if (multiplies (visited, frequency))
recomputed_scores_PR;
link:=dequeue_top_link(frontier);
doc:=fetch(link);
score_sim:=sim(topic, doc);
enqueue (buffered_pages, doc, score_sim);
if (#buffered_pages >= MAX_BUFFER)
dequeue_bottom_links(buffered_pages);
merge(frontier, extract_links(doc), score_PR);
if(#frontier > MAX_BUFFER)
dequeue_bottom_links(frontier);
}
}
	Starting_urls=Search Engine(topic_keywords);
	For each link in starting_urls
	{
	Linkscore=sim(topic,url)+sim(topic, anchor_text);
	Enquence(frontier, link, Linkscore);
	}
	While (visited < MAX_PAGES)
	{
	if (multiplies (visited, frequency))
	recomputed_scores_To-PR;
	link:=dequeue_top_link(frontier);
	doc:=fetch(link);
	score_sim:=sim(topic, doc);
	enqueue (buffered_pages, doc, score_sim);
	if (#buffered_pages >= MAX_BUFFER)
	dequeue_bottom_links(buffered_pages);
	merge(frontier, extract_links(doc), score_To-PR);
	if(#frontier > MAX_BUFFER)
	dequeue_bottom_links(frontier);
	}

Note that the algorithm has to maintain a data structure of visited pages in addition to the frontier, in order to compute score_To-PR(j). This buffer can contain at most MAX BUFFER

3.3.3 HITS Algorithm

HITS stands for Hypertext Induced Topic Search [12]. Unlike PageRank which is a static ranking algorithm, HITS is search query dependent. When the user issues a search query, HITS first expands the list of relevant pages returned by a search engine and then produces two rankings of the expanded set of pages, authority ranking and hub ranking.

An authority is a page with many in-links. The idea is that the page may have good or authoritative content on some topic and thus many people trust it and link to it. A hub is a page with many out-links. The page serves as an organizer of the information on a particular topic and points to many good authority pages on the topic. When a user comes to this hub page, he/she will find many useful links which take him/her to good content pages on the topic. The key idea of HITS is that a good hub points to many good authorities and a good authority is pointed to by many good hubs. Thus, authorities and hubs have a mutual reinforcement relationship. Figure 3.4 shows a set of densely linked authorities and hubs.

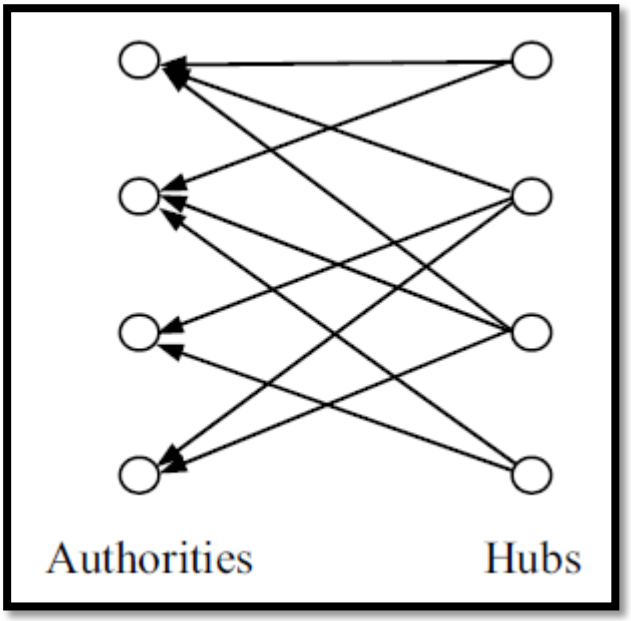


Figure 3.4: A densely linked set of authorities and hubs [12]

Hubs and authorities are assigned respective scores. Let a_p and h_p represent the authority and hub scores of page p, respectively. B(p) and F(p) denote the set of

referrer and reference pages of page p, respectively. The scores are computed in a mutually reinforcing way and the algorithm iteratively proceeds as follows:

$$a_p = \sum_{q \epsilon B(p)} h_q$$

$$h_p = \sum_{q \epsilon F(p)} a_q$$

3.3.3.1 Strengths and Weaknesses of HITS

The main strength of HITS [12] is its ability to rank pages according to the query topic, which may be able to provide more relevant authority and hub pages. The ranking may also be combined with information retrieval based rankings. However, HITS has several disadvantages.

- First of all, it does not have the anti-spam capability of PageRank. It is quite easy to influence HITS by adding out-links from one's own page to point to many good authorities. This boosts the hub score of the page. Because hub and authority scores are interdependent, it in turn also increases the authority score of the page.
- Another problem of HITS is topic drift. In expanding the root set, it can easily collect many pages (including authority pages and hub pages) which have nothing to do the search topic because out-links of a page may not point to pages that are relevant to the topic and in-links to pages in the root set may be irrelevant as well because people put hyperlinks for all kinds of reasons, including spamming.
- The query time evaluation is also a major drawback. Getting the root set, expanding it and then performing eigenvector computation are all time consuming operations.

3.3.4 Fish Search Algorithm

The difference between a good personalized search engine and existing search engine lies in: instead of collecting and searching all the website could be visited, the search is limited within the most relative links required by user, avoid visiting the irrelative part in the Internet. "Fish-search" [5] is one of the famous dynamic search algorithm

that capitalizes on the intuition that relevant documents often have relevant neighbours. Thus, it searches deeper under documents that have been found to be relevant to the search query, and stops searching in "dry" areas.

Fish-search algorithm treats Internet as a directed graph, webpage as node and hyperlink as edge, so the search operation could be abstracted as a process of traversing directed graph. For every node we judge whether it is relative, 1 for relevant, 0 for irrelevant. Fish-search algorithm maintains a list, which keeps URL of page to be searched. The URLs have different priority, the URL with more superior priority will be located at the front of the list, and will be searched sooner than others. If relative page is found, it stands for that the food has been found by the fish, and more healthy reproduction, the same as more relative links covered by the page. The key point of Fish-search algorithm lies in the maintenance of URL's order. Based on the value of potential-score, Fish-search algorithm changes the order in the list. The algorithm works as follows:

- **Get** as Input parameters, the initial node, the width (width), depth (D) and size (S) of the desired graph to be explored, the time limit, and a search query
- **Set** the depth of the initial node as depth = D, and **Insert** it into an empty list
- **While** the list is not empty, and the number of processed nodes is less than S, and the time limit is not reached
    1. **Pop** the first node from the list and make it the current node
    2. **Compute** the relevance of the current node
    3. **If** depth > 0:
        1. **If** current_node is irrelevant

           Then
            - **For** each child, child_node, of the first width children of current_node
                - **Set** potential_score(child_node) = 0.5
            - **For** each child, child_node, of the rest of the children of current_node
                - **Set** potential_score(child_node) = 0

Else

- **For** each child, child_node, of the first (α * width) children of current_node (where α is a pre-defined constant typically set to 1.5)
  - **Set** potential_score(child_node) = 1
- **For** each child, child_node, of the rest of the children of current_node
  - **Set** potential_score(child_node) = 0

2. **For** each child, child_node, of current_node,
   - **If** child_node already exists in the priority list, **Then**
     1. **Compute** the maximum between the existing score in the list to the newly computed potential score
     2. **Replace** the existing score in the list by that maximum
     3. **Move** child_node to its correct location in the sorted list if necessary

     Else

     Insert child_node at its right location in the sorted list according to its potential_score value

For each child, child_node, of current_node,

Compute its depth, depth(child_node), as follows:

0. **If** current_node is relevant,
   Then Set depth(child_node) = D
   Else depth(child_node) =depth(current_node) – 1
1. **If** child_node already exists in the priority list **Then**

1. **Compute** the maximum between the existing depth in the list to the newly computed depth
2. **Replace** the existing depth in the list by that maximum.

End While

Figure 3.5: Fish Search Algorithm [5]

3.3.4.1 Limitations of Fish Search Algorithm

It assigns a relevance score in a discrete manner (1 for relevant, 0 or 0.5 for irrelevant) using primitive string- or regular-expression match. More generally, the key problem of the fish-search algorithm is the very low differentiation of the priority of pages in the list. When many documents have the same priority, and the crawler is restricted to a fairly short time, arbitrary pruning occurs - the crawler devotes it's time to the documents at the head of the list. Documents which are further along the list whose scores may be identical to some further along may be more relevant to the query. In addition, cutting down the number of addressed children by using the width parameter is arbitrary, and may result in loosing valuable information. Also the fish-search algorithm didn't make full use of the following three rules of flocking behaviour [22].

- Collision Avoidance**:** avoid collisions with nearby flock-mates.
- Velocity Matching**:** attempt to match velocity with nearby flock-mates.
- Flock Centering**:** attempt to stay close to nearby flock-mates.

In recent years there is a lot of work is done on fish search algorithm to overcome its limitations. In improved fish search algorithm "distance" parameter is introduced to measure the distance between fish. Two fish moving in different direction can be regarded as two directed graphs. We can control the search direction with the "distance" between the two directed graphs, the "distance" stands for the searching range where the two fish had searched. There should be reasonable distance between the two fish, neither too close nor too far; too close will lead to crossover repeated search, while too far will make the search algorithms run slowly because of the wide searching range. In order to control the searching time and range efficiently, we could

set the "distance" between the two fish. The "distance" of the two directed graphs can be regarded as the distance between the centers of two directed graphs.

3.3.5 Shark Search Algorithm

Considering the limitations of the Fish Search algorithm as stated above, a powerful improved version of Fish Search algorithm is developed known as Shark Search [2]. In this algorithm, One immediate improvement is that instead of the binary (relevant/irrelevant) evaluation of document relevance, it returns a "fuzzy" score, i.e., a score between 0 and 1 (0 for no similarity whatsoever, 1 for perfect "conceptual" match) rather than a binary value.

These heuristics are so effective in increasing the differentiation of the scores of the documents on the priority list, that they make the use of the width parameter redundant, there is no need to arbitrarily prune the tree. Therefore, no mention is made of the width parameter in the shark-search algorithm that is more formally described in Fig.3.6.In the fish-search algorithm, replace step for computing the child's potential score, with the following:

1. **Compute** the inherited score of child_node, inherited_score(child_node), as follows:
    - **If** relevance(current_node) > 0 (the current node is relevant)
      Then inherited_score(child_node) = δ * sim(q,current_node)
      where δ is a predefined decay factor.
      Else inherited_score(child_node) = δ * inherited_score(current_node)
2. **Let** anchor_text be the textual contents of the anchor pointing to child_node, and anchor_text_context, the textual context of the anchor (up to given predefined boundaries)
3. **Compute** the relevance score of the anchor text as anchor_score = sim(q,anchor_text)
4. **Compute** the relevance score of the anchor textual context as follows:
    - **If** anchor_score > 0,
      Then anchor_context_score = 1
      Else anchor_context_score = sim(q,anchor_text_context)

5. **Compute** the score of the anchor, that we denote neighbourhood_score as follows:

   neighbourhood_score = β* anchor_score + (1 – β) * anchor_context_score

   where β is a predefined constant.

6. **Compute** the potential score of the child as -

   potential_score (child_node) = γ * inherited_score(child_node) + (1 – γ) * neighbourhood_score(child_node)

   where γ is a predefined constant

Figure 3.6: Shark Search Algorithm [2]

Chapter 4

Problem Statement

From the beginning, a key motivation for designing Web crawlers has been to retrieve Web pages and add them or their representations to a local repository. Such a repository may then serve particular application needs such as those of a Web search engine. Due to the size of the Web and its constant change, it is basic to fetch first relevant pages before the irrelevant ones in order to not waste resources and time.

The focused crawling strategy based on the intuition that relevant pages often contain relevant links. It searches deeper when relevant pages are found, and stops searching at pages not as relevant to the topic. Unfortunately, this traditional method of focused crawling shows an important drawback when the pages about a topic are not directly connected.

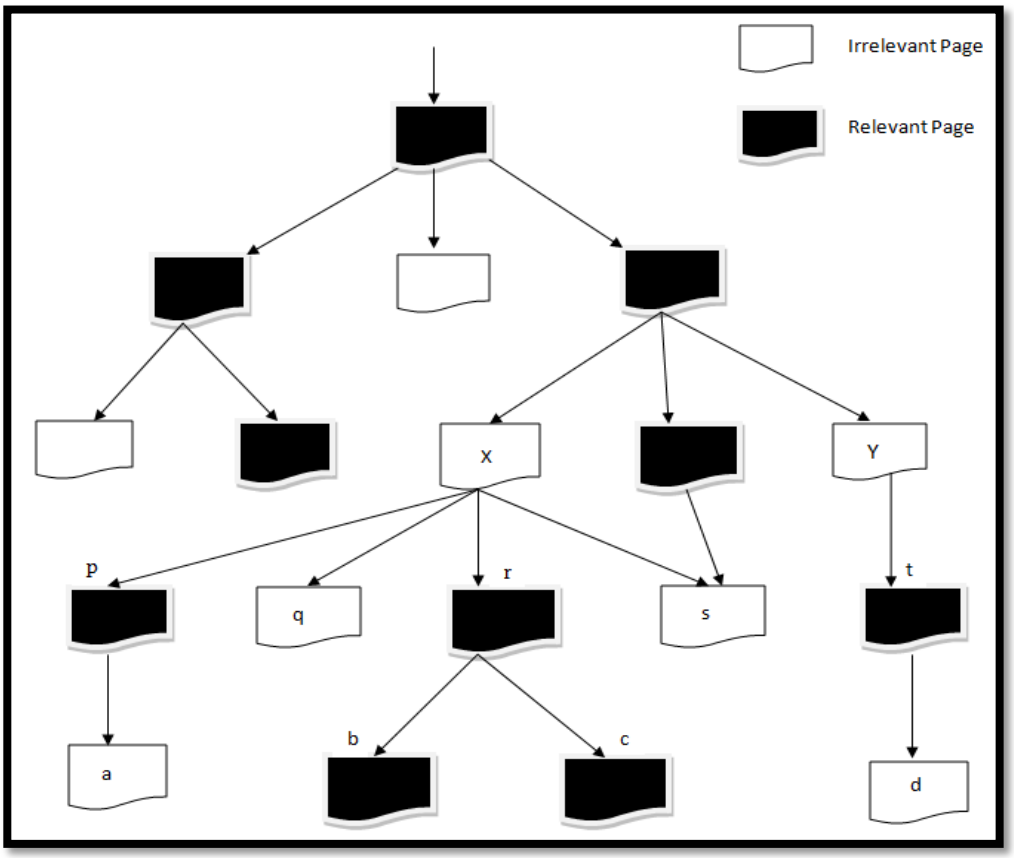


Figure 4.1: An Example Structure of Web Pages

Web pages relevant to the desired domain could be separated by irrelevant pages. Because focused crawlers using local search algorithms will give up searching when they encounter irrelevant pages, they will not be able to explore relevant Web pages which are separated from the initial pages containing the starting URLs. Figure 4.1 illustrates the problem described above. From fig4.1 we can see that at level 2 there are some irrelevant pages (X,Y) which are discarding relevant pages (p, r, t) at level 3 and (b, c) at level 4 from the crawling path. As a solution of this problem this thesis gives an algorithm that allows the crawler to follow several bad pages in order to get a good page. The working principle of this algorithm is to go on crawling up to a given max Level from the irrelevant page.

Objectives:

- Study of the existing Crawler algorithms
- Identifying inefficiency in focused crawling algorithms (For example missing relevant pages which are connected by irrelevant pages)
- Design of an algorithm that allows the crawler to follow several irrelevant pages in order to get a relevant page
- Overcoming inefficiency in focused crawling by implementing an algorithm with the help of an example graph structure which is tacking as input

Chapter 5

Algorithms & Implementation

A number of crawling strategies are illustrated in the previous chapters. In this thesis some of the crawling algorithms are implemented using programming in C and the results are obtained by running these algorithms on a same graph structure.

5.1 Data Structures and Programming Language used

This section, describes the data structures and the programming language used in the implementation of algorithms and start with giving introduction to C language which is used as main language in implementation.

5.1.1 Introduction to C language

C is a general-purpose, block structured, procedural, imperative computer programming language developed in 1972 by Dennis Ritchie at the Bell Telephone Laboratories for use with the UNIX operating system. It has since spread to many other platforms. Although C was designed as a system implementation language, it is also widely used for applications. C has also greatly influenced many other popular languages, especially C++, which was originally designed as an extension to C.

5.1.2 Linked List Representation of Graph

A linked list is a collection of objects linked to one another like a string of pearls. As a minimum, a linked list knows about its first element, and each element knows about its successor. The structure of linked list is shown in figure 5.1

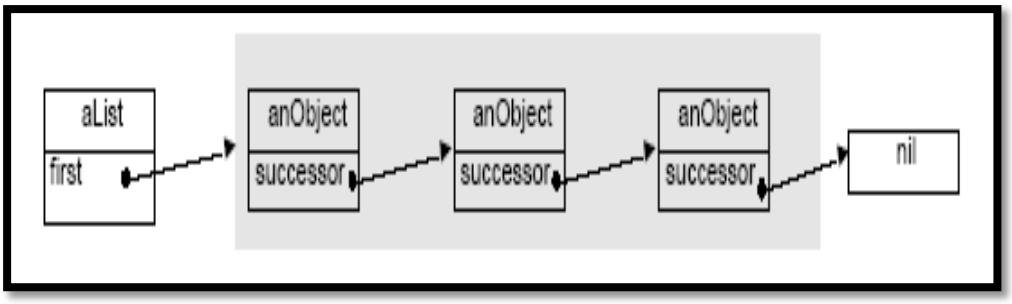


Figure 5.1: Linked List Representation

As mentioned in the previous chapters, the structure of World Wide Web is Graphical in nature. Web graphs contain a few hundred million vertices and a few billion edges. With each vertices representing a web page and each edge representing link from one page to another web-pages.

So, to implement these algorithms, a technique of representing Graph in memory i.e. linked list Structure is used. Linked list is a collection of nodes and in C language; the node structure can be given as below:

```
                struct node
{
                int vertex;
                float weight;
                node *link;
};
```

Figure 5.2: Node Structure of Graph in C

5.2 Assumptions

During the implementation process, some assumptions are taken in to account just for simplifying algorithms implementation and results.

As mentioned in Chapter 3, the basic procedure executed by any web crawling algorithm takes a list of seed URLs as its input and repeatedly executes the following steps:

i. Remove a URL from the URL list
ii. Determine the IP address of its host name
iii. Download the corresponding document
iv. Check the Relevance of Document
v. Extract any links contained in it
vi. Add these links back to the URL list

In this implementation of algorithms, some assumptions are made related to steps (iv) and (v) of the above procedure.

5.2.1 Relevancy Calculation

In crawling algorithms, the relevancy is calculated by parsing the web page and matching its contents with the desired key-words. After matching, a Relevancy Value corresponding to that page is generated by the Relevancy Calculator component.

These implementations, mainly check how the traversing is done by various algorithms, not the concept of parsing as it in itself is a different module. The parsing module can be simply replaced by implementation source code. For the time being, to calculate relevancy of different documents, random function is used to generate relevancy values between 0 and 1. Whenever a page is traversed, a random relevancy is generated corresponding to it.

5.2.2 Link Extraction

As the implementation of parser is not available, the link extraction from a page is performed by traversing its adjacency list, and adding the pages corresponding to that list into the unvisited URL list.

5.2.3 Input Graph for Implementation

Since the entire web can't be selected for implementation, a graph structure is used as shown below to implement all the algorithms on it.

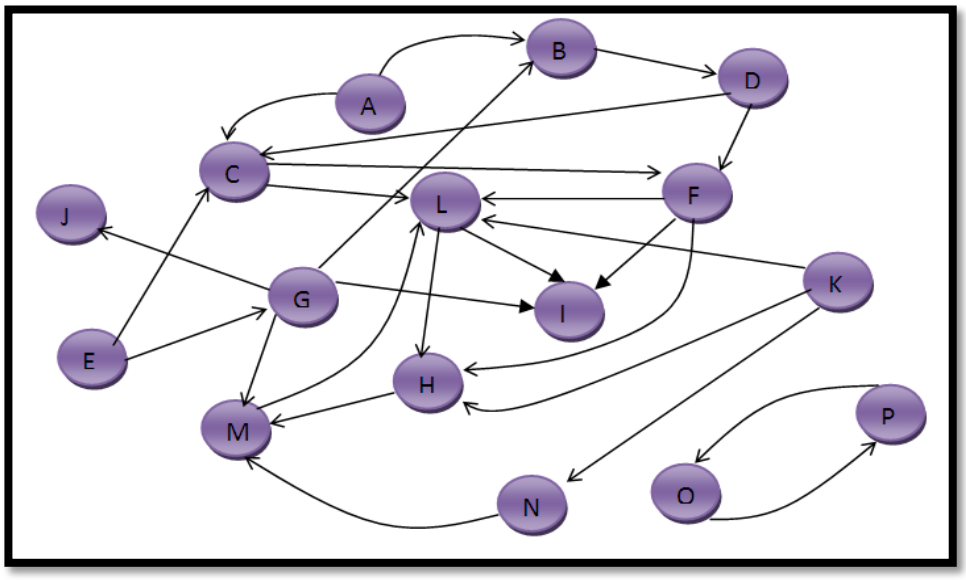


Figure 5.3: Input Graph for Implementation

As for example, these 16 nodes represent 16 web-pages, while each edge represents link between two web-pages. As in real web, some web-pages are not having links with others; nodes O and P represent this case. Programs which are made in C language are executed on this dummy web structure and corresponding results will be noted.

5.3 Pseudo-Code of Algorithms

Four crawling algorithms in C language are implemented. The Pseudo-code of these crawling algorithms is given below:

5.3.1 Pseudo Code of Breadth First Algorithm

The pseudo-code of a Breadth First Approach is as follows:

Breadth_First_Algo(seed URLs) // Start with given Seed URLs as input

Insert_Frontier (seed URLs); // Insert seed URLs into the Frontier
While (Frontier! = Empty) // Crawling Loop

Link: =Remove_Frontier (URL); // Pick new link from the frontier
Webpage: = Fetch (Link); // Open Webpage corresponding to selected link
Insert_Frontier (newly_extracted_links); // Add new links into frontier

End While Loop

Figure 5.4: Pseudo-code of Breadth-First Approach

The working of code is simple. This algorithm start with a set of URLs, called seed and insert these seed into queue, also called Frontier. Crawling process continues

while queue is not empty, every time a new URL is removed from the front of queue and is traversed. All new links found in this page are inserted into the queue.

5.3.2 Pseudo- Code of Naive Best First Algorithm

Pseudo-code for a Best First Approach can stated as follows:

Best_First_Algo(seed URLs) // Start with given Seed URLs as input
{
Insert_Frontier (seed URLs, 1); // Insert seed URLs
Repeat 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
Repeat For each child_node of Webpage
{
Rel_val (child_node):= Relevancy(desired_topic, webpage); // Calculate Rel_val
Insert_Frontier (child_node, Rel_val); // Add new links with Rel Val into frontier
}
End While Loop

Figure 5.5: Pseudo- Code of Naive Best First Approach

This algorithm starts with a set of seed URLs; initially the relevancy value is 1 for all seed URLs. These URLs are added to the frontier. Priority queue is used to implement the Frontier of any Best First Algorithm. Page is fetched and new links (child nodes are fetched) are assigned a relevancy value according to their relevance with desired keywords. This relevancy value is assigned to the newly found links in it, and these newly found URLs are inserted into the frontier along with their respective relevancy values. Again the URL with highest relevancy is picked from the frontier and this procedure goes on adding more and more new pages into the frontier.

5.3.3 Pseudo-Code of Fish Search Algorithm

```
Fish Search ( Initial node(seed) , depth (D) , the time limit, search query)
{
Set Depth of the initial node = D, and Insert into an empty list;
Repeat While (Frontier! = Empty)
current_node:= First_Frontier(URL);
Webpage: = Fetch (Current_Node);    // Open Webpage corresponding to selected link
Rel_val := Relevancy(desired_topic, current_node);    // calculate relevancy(0 or 1)
If depth(current_node) > O
{
If (current_node is relevant)          // i.e. relevancy of current_node is 1

Repeat For (each child_node of current_node)
{
Insert_Frontier (child_node);
Set potential_score(child_node):=1;
Set depth(child_node)=D;
}

Else                                    // if current_node is irrelevant
 Repeat For (each child_node of current_node)
{
Insert_Frontier (child_node, last);        // insert all children into the last
Set potential_score(child_node)= 0;
depth(child node)=depth(current_node)- 1;     // decrease the searching depth
}
End While Loop
```

Figure 5.6: Pseudo-code of Fish Search Approach

5.3.4 Input Graph for Dealing with Irrelevant Pages

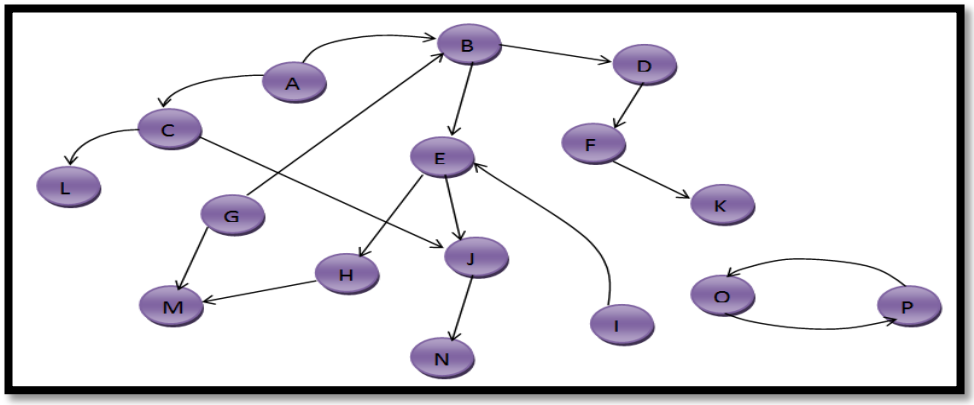


Figure 5.7: Input Graph for Dealing with Irrelevant Pages

5.3.4 Pseudo-code of Dealing with Irrelevant Pages

```
if (page is Irrelevant)
{
   Initialize level = 0;
   url_list = extract_urls(page);// extract all the urls from the page
     for each u in url_list
     {
       compute LinkScore(u) using equation (3);
       IrrelevantTable.insert(u, LinkScore(u), level);
        // insert u into irrelevant table with linkscore and level value
     }
   reorder IrrelevantTable accord to LinkScore(u);
   While ( IrrelevantTable.size > 0)
   {
      get the url with highest score and call it Umax ;
      if ( Umax.level < = maxLevel)
      {
         page = downloadPage(Umax);// download the URL Umax
         calculate relevance of the page using equation (1);
         if ( page is relevant)
         {
           RelevantPageDB = page;// put page into relevant page database
           if ( Umax.level < maxLevel)
           {
              level ++ ;
              url_list = extract_urls(page);
              for each u in url_list
              {
                compute LinkScore(u) using equation (3);
                IrrelevantTable.insert(u, LinkScore(u), level);
              }
              reorder IrrelevantTable accord to LinkScore(u);
           }
         }
         else
         {
            for each u in IrrelevantTable
            {
               if (LinkScore(u) < = LinkScore(Umax)
                 && u.level = = Umax.level)
               u.level ++;
            }
          }
    }
 }}
```

Figure 5.7: Pseudo-code of Dealing with Irrelevant Pages

Chapter 6

Implementation Results

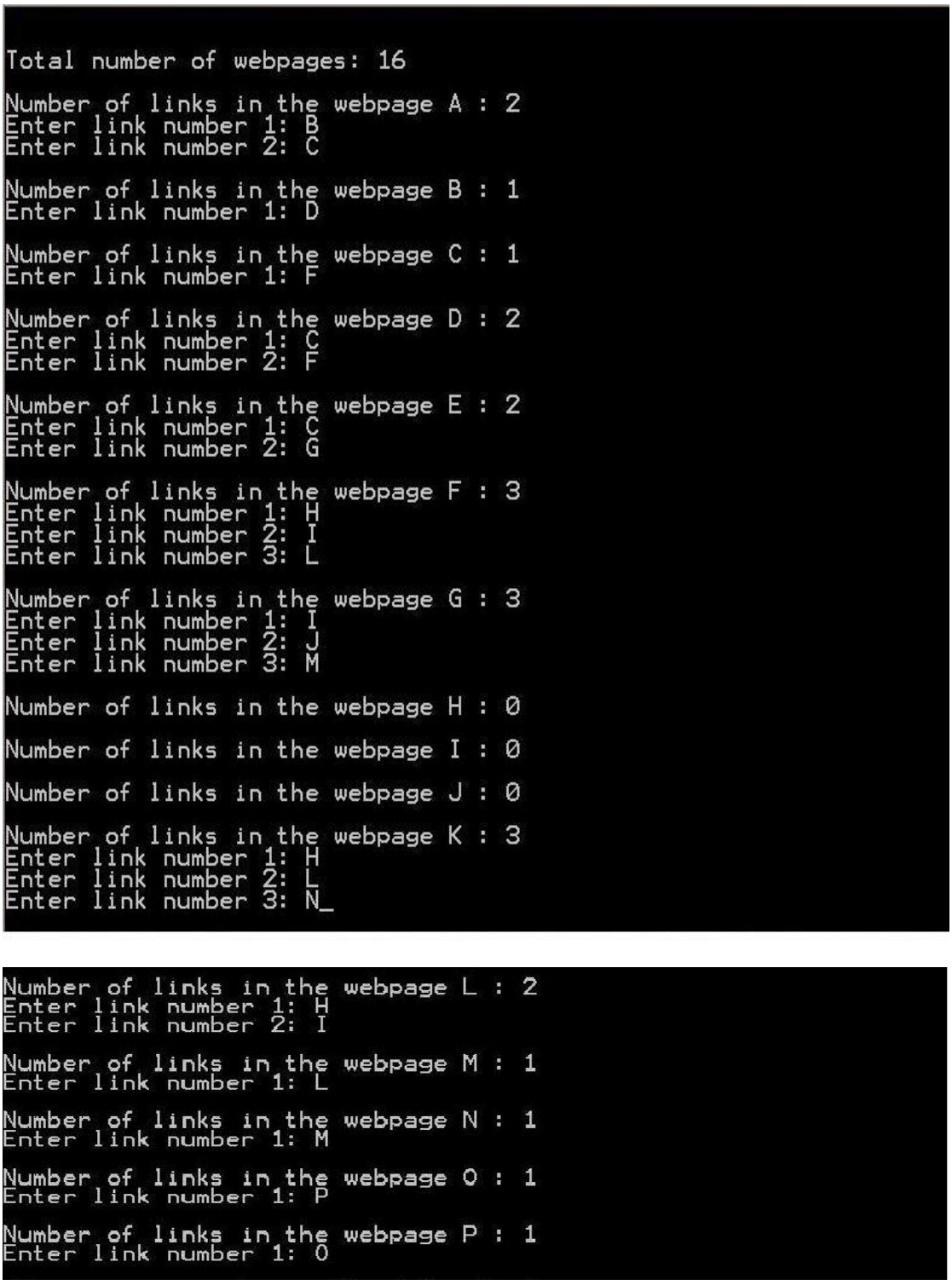
After implementing the Pseudo-codes given in the previous section, the following results are achieved. Referring to figure 5.3, the input graph is used and various results are noted according to input.

Table 6.1: Adjacency Table of input Graph

Page number	Node Label	Approachable Pages
1	A	B , C
2	B	D
3	C	F
4	D	C , F
5	E	C , G
6	F	H , I , L
7	G	I , J , M
8	H	-
9	I	-
10	J	-
11	K	H , L , N
12	L	H , I
13	M	L
14	N	M
15	O	P
16	P	O

6.1 Input of Graph

The given graph is input into the memory using its Adjacency Table as shown in Figure 6.1. The input of the graph is shown in the following screenshot:

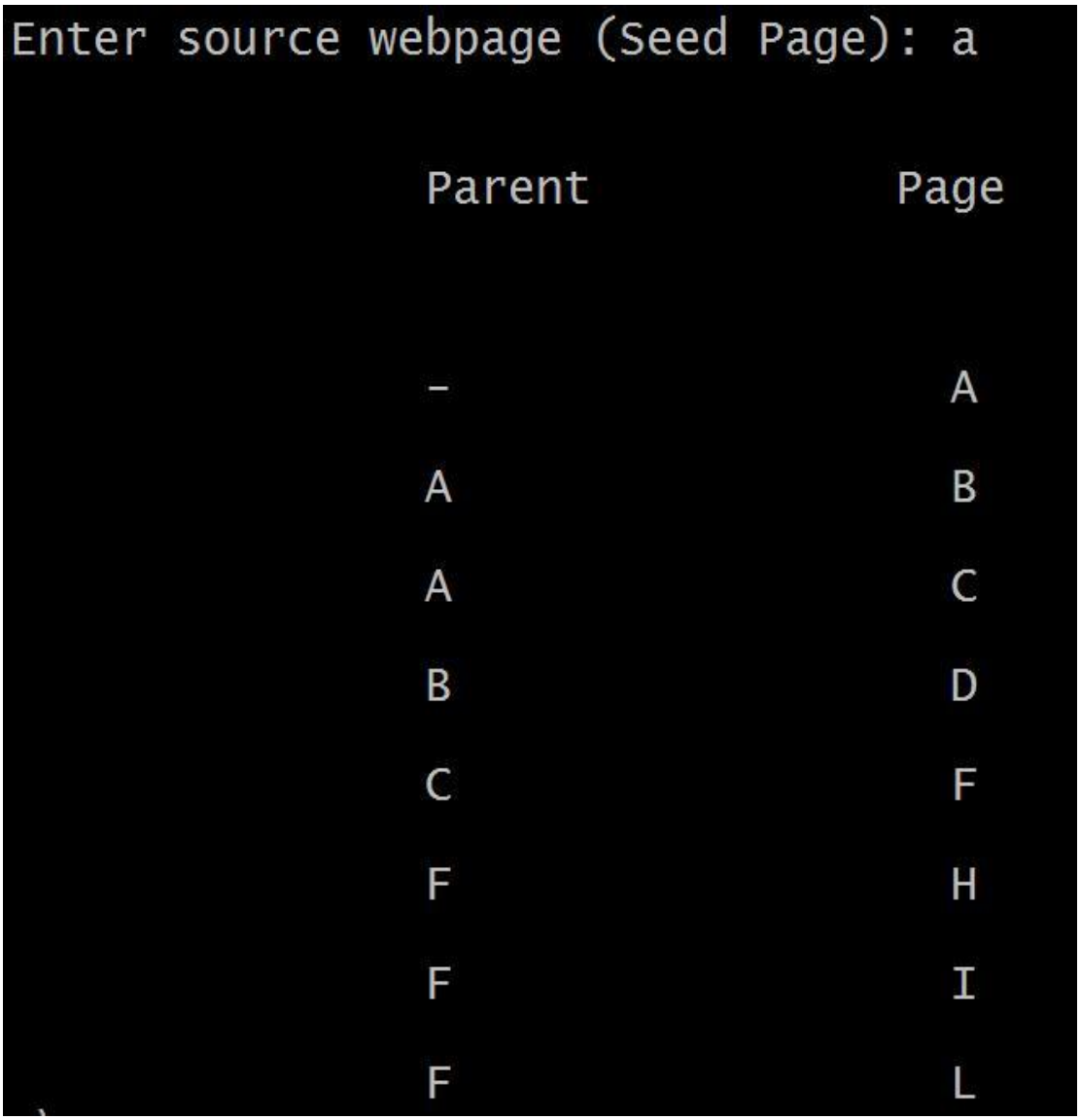


Screenshot 6.1: Input of Graph along with Adjacency Table

Thus in this way, the entire graph is input in to the memory along with its adjacency Table. Now execute crawling programs on this input and the corresponding results for each program are noted.

6.2 Output Results for Breadth-First Search

When node A is selected as starting seed, the following nodes are approached by the crawler along with their sequence.



Screenshot 6.2: Output result for BFS

The working of any Breadth-First algorithm is very simple. It simply works of first come first serve. Algorithm starts with page A. After processing node A, its child nodes B and C are inserted into the frontier. Again the next page is fetched from the

frontier and is processed, its children inserted into frontier and so on. This procedure continues until the frontier gets empty. Breadth-First is used here as a baseline crawler; since it does not use any knowledge about the topic, and its performance is considered to provide a lower bound for any of the more sophisticated algorithms.

6.3 Output Results for Naive Best First Search

The same input graph when uses the Naive Best First approach for crawling, the following output results:

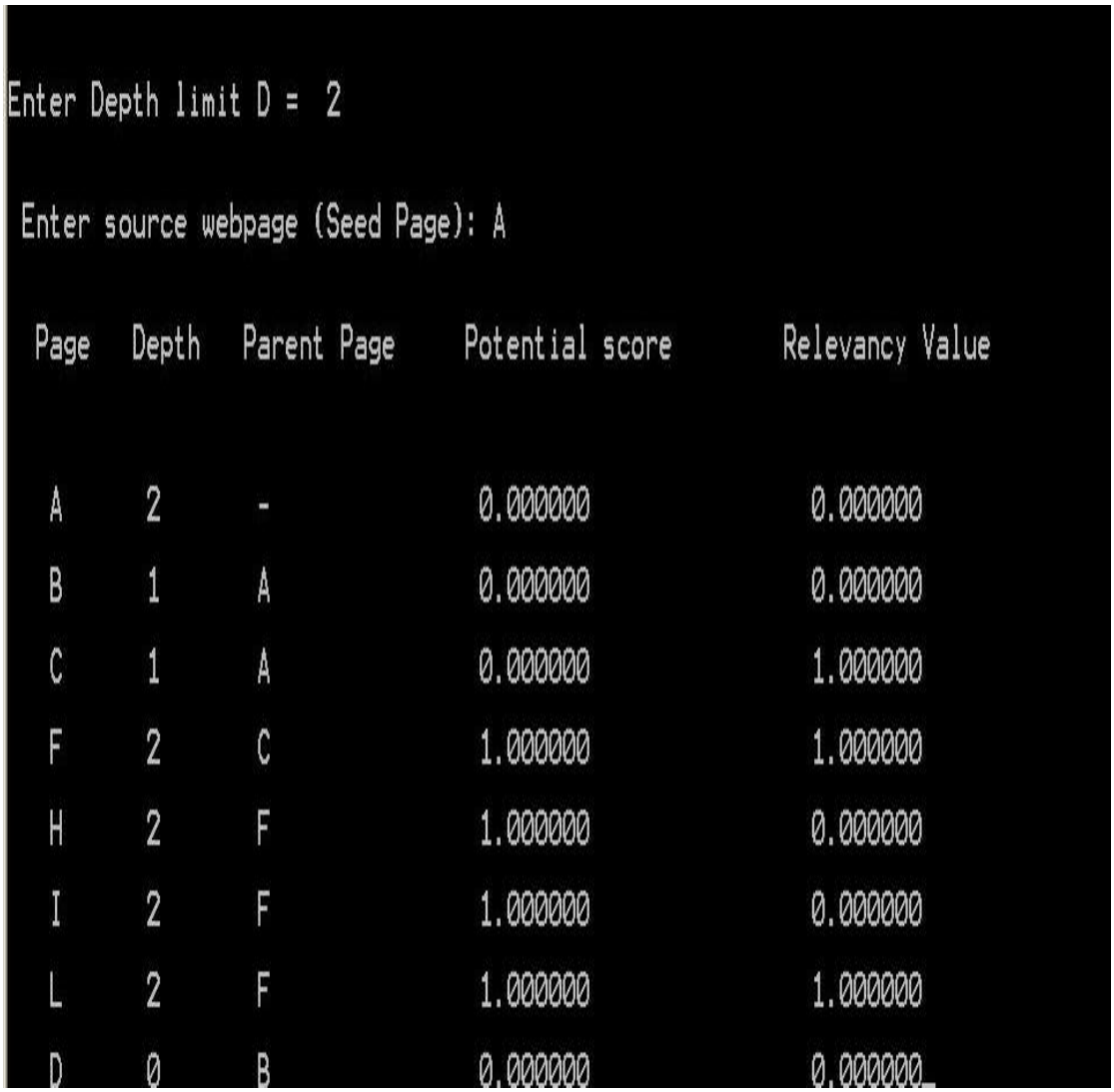
```
Number of links in the webpage P : 1
Enter link number 1: o

 Enter source webpage (Seed Page): a

       Parent            Relevancy of Page          Page Page

        -                 0.500000                   A

        A                 0.100000                   B

        B                 0.500000                   D

        D                 0.300000                   F

        F                 1.000000                   L

        F                 0.600000                   I

        F                 0.300000                   H

        A                 0.100000                   C
```

Screenshot 6.3: Output Results for Naive Best First approach

Note that in naive best first, the preference of next page to be approached depends upon the relevancy of that page. This output screenshot depicts the idea behind working of any best first approach. Traversing starts from page A. Now the relevancy of A's children B and C comes out to be 0.1 and 0.1 respectively. Along with respective relevancies, B and C are inserted in frontier. Every time, the page with highest relevancy value is picked from the frontier.

6.4 Output Results for Fish Search

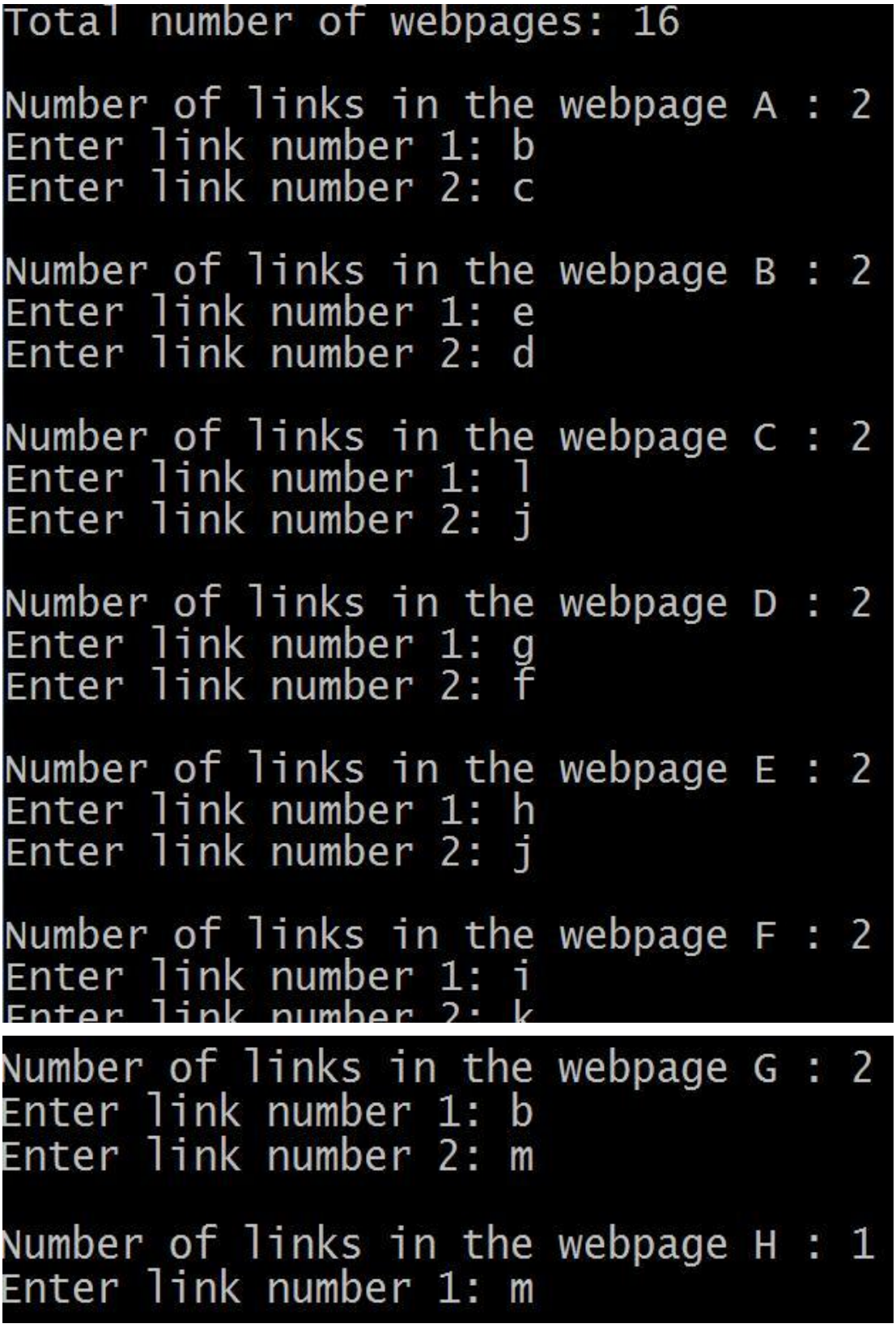
A very good and efficient approached as discussed in previous chapters is – Fish Search given by De Bra. It is used to traverse up to some pre-defined depth. Also the relevancy is checked along with going into the depth of the graph. The output screen for Fish Search is shown below:



Screenshot 6.4: Output of Fish Search approach

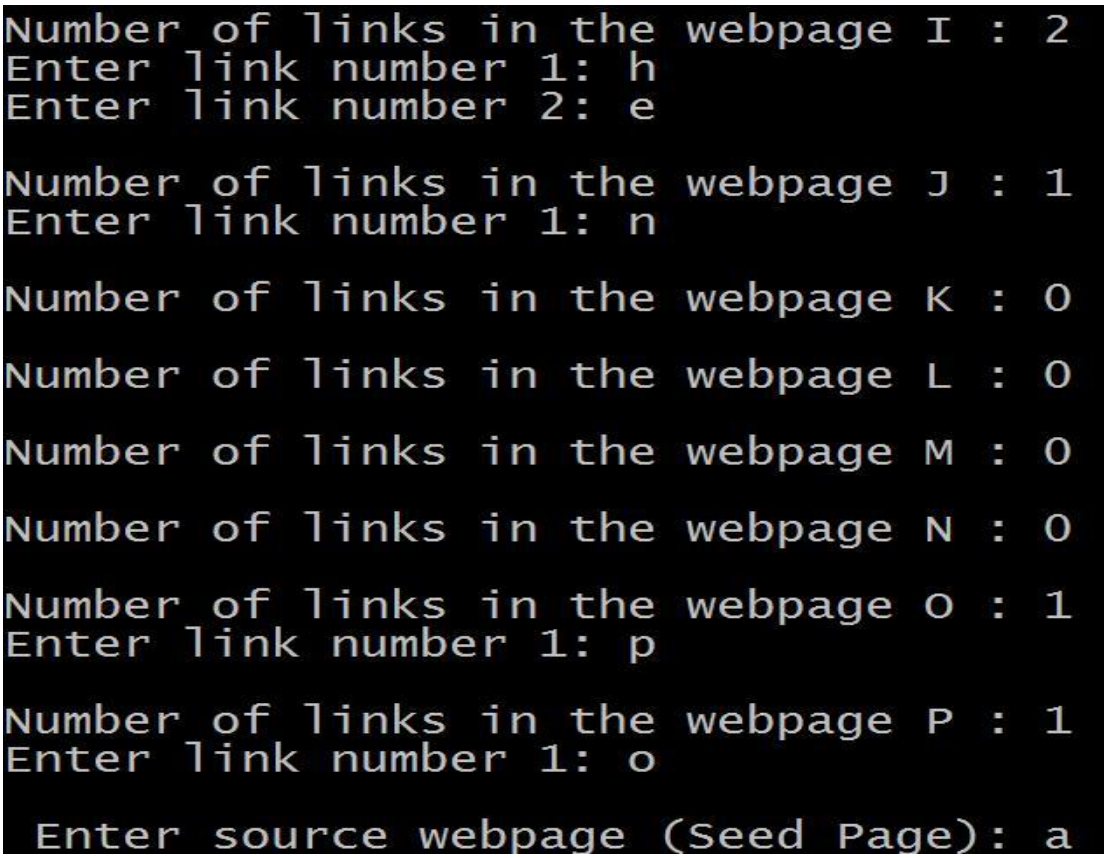
Algorithm start with Depth =2, check relevancy of A, its 0. So the potential score of its children B and C is set to 0 and the depth is set 1 for both. Whenever a page is relevant, the potential score of its children is set to 1 and depth is set to 2. Otherwise, potential score of children is set to 0 and depth is decremented by 1. As soon as depth reaches 0, it stops traversing.

6.5 Input Graph for Dealing with Irrelevant Pages



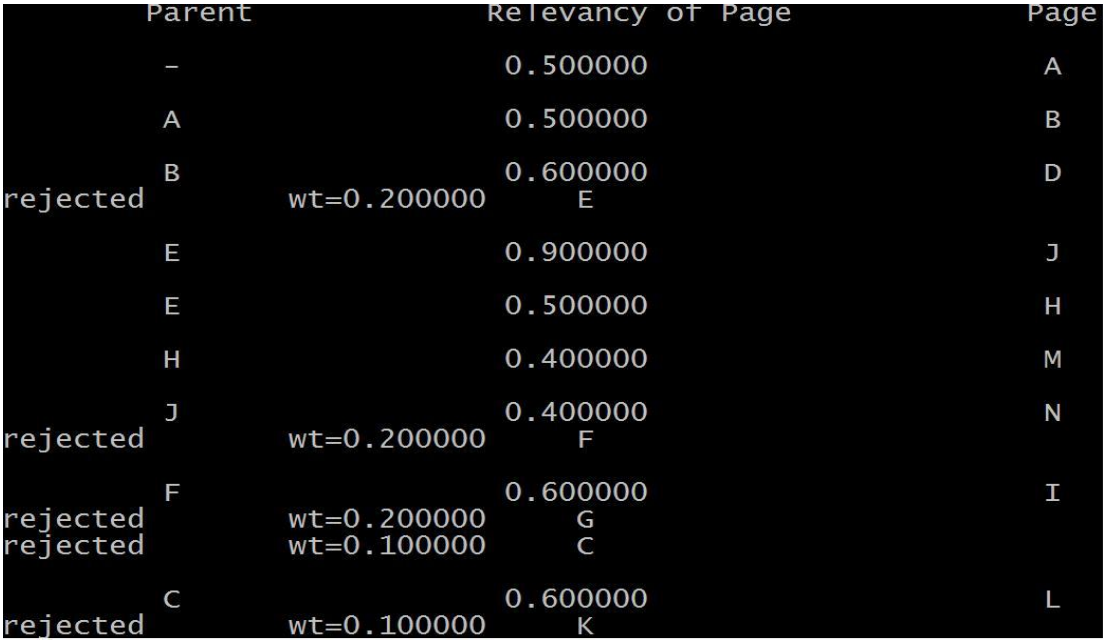
Screenshot 6.5 Input of Graph for Dealing with Irrelevant Pages

Input of Graph for Dealing with Irrelevant Pages (continued...)



Screenshot 6.6 Input of Graph for Dealing with Irrelevant Pages (continued...)

6.6 Output Results for Dealing with Irrelevant Pages



Screenshot 6.7: Output of Dealing with irrelevant pages

As cleared from the output screenshot 6.7, the relevant pages L, M, H, N, I, J are obtained even though these are connected by the pages C, E, F, G and K which are irrelevant.

Chapter 7

Conclusion & Future Work

7.1 Conclusion

In this thesis, briefly discussed about Search Engines, Basics of Crawling and 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. This thesis, presented a method for focused crawling that allows the crawler to go through several irrelevant pages to get to the next relevant one when the current page is irrelevant. From the experimental results, proposed approach has given better performance than the BFS crawler.

7.2 Future Work

Although the initial results are encouraging, there is still a lot of work to do for improving the crawling efficiency. A major open issue for future work is to do more extensive test with large volume of web pages. Future work also includes code optimization and URL queue optimization, because crawler efficiency is not only depends to retrieve maximum number of relevant pages but also to finish the operation as soon as possible.

References

[1] Brin S, and Page L, "The Anatomy of a Large-Scale Hyper textual Web Search Engine", In Proceedings of the 7th International World Wide Web Conference, Brisbane, Australia, Apr1998, 14 –18. pp. 107–117.

[2] Hersovici M, Jacovi M , Maarek Y.S , Pelleg D , Shtalhaim M , and Ur S, "The Shark-Search Algorithm - An Application: Tailored Web Site Mapping", Computer Networks and ISDN Systems, 1998, Vol. 30, No 1-7, pp. 317-26.

[3] Chakrabarti S, van den Berg M and Dom B, "Focused Crawling: A New Approach for Topic Specific Resource Discovery", In Proceedings of the 8th International World Wide Web Conference (WWW8) 1999.

[4] Salton G, Wong A and Yang C.S, "A Vector Space Model for Automatic Indexing", Communications of the ACM, 1975, 18(11):613–620.

[5] De Bra P, Houben G , Kornatzky Y, and Post R, "Information Retrieval in Distributed Hypertexts", Proceedings of RIAO'94, Intelligent Multimedia, Information Retrieval Systems and Management, New York, 1994, pages 481–491.

[6] Menczer F, Pant G and Srinivasan P, "Topical Web Crawlers: Evaluating Adaptive Algorithms", ACM Transactions on Internet Technology (TOIT), Nov. 2004, 4(4):378–419.

[7] Kraft R and Stata R, "Finding buying guides with a web carnivore", First Latin American Web Congress (LA-WEB'03), 2003, pages 84–92.

[8] Pant G, Tsjoutsiouliklis K, Johnson J and Giles C. L, "Panorama: Extending digital libraries with topical crawlers", Proceedings of the 4th ACM/IEEE-CS Joint Conference on Digital Libraries, 2004, p. 142–150.

[9] McCown F and Nelson M, "Agreeing to Disagree: Search Engines and their Public Interfaces", ACM IEEE Joint Conference on Digital Libraries (JCDL) Vancouver, British Columbia, Canada, June 17-23, 2007 , p. 309-318.

[10] Chang C, Kayed M, Girgis MR and Shaalan KF , "A Survey of Web Information Extraction Systems", IEEE Transactions on Knowledge and Data Engineering, TKDE-0475-1104.R3, 2006.

[11] "Inner work of Crawler/Robot", available at: http://www.webreference.com/authoring/robots/

[12] Bing liu, "web data mining", from Chapter 6, 7, 8, Springer-Verlag Berlin Heidelberg, 2007, pages183-235,237-270,273-318.

[13] Soumen Chakrabarti, "Mining the Web", from Chapters 2, 3, giga-pedia, pages17-77.

[14] Porter M.F, "An Algorithm for Suffix Stripping", Program 1980, Vol. 14(3), pp. 130- 137.

[15] Menczer F, Pant G, Ruiz M and Srinivasan P, "Evaluating topic-driven Web crawlers", In Proc. 24th Annual Intl, ACM SIGIR Conf. on Research and Development in Information Retrieval, 2001.

[16] Menczer F, Pant G and Srinivasan P, "Crawling the Web" available at: http://dollar.biz.uiowa.edu/~pant/Papers/crawling.pdf .

[17] Yang Yongsheng, Wang Hui, "Implementation of Focused Crawler", COMP 630D Course Project Report, 2000.

[18] Najork M, Winnre J. L, "Breadth-first search crawling yields high quality pages", In Proceedings of the 10th International World Wide Web Conference 2001.

[19] Cho J, Garcia-Molina H, Page L, "Efficient crawling through URL ordering", Comput.Netw30, 1998, Pages 1–7, 161–172.

[20] Rafeal Romero Trujilo, "Simulation tool to study focused web crawling strategies", Exjobreport, March 2006.

[21] Yulian Zhang**,** Chunxia Yin**,** Fuyong Yuan "An Application of Improved PageRank in Focused Crawler", Fourth International Conference on Fuzzy Systems and Knowledge Discovery (FSKD)2007 IEEE.

[22] Fang-fang Luo, Guo long Chen, Wen-zhong Guo, “An Improved Fish-search Algorithm for Information Retrieval” Proceeding of NLP-KE 2005.

[23] Xianchao, Zhang, Hong Yu, Cong Zhang and Xinyue Liu, “An Improved Weighted HITS Algorithm Based on Similarity and Popularity”**,** Second International Multisymposium on Computer and Computational Sciences (SIMCCS), IEEE 2007.

[24] “Implement advanced search with Lucene”, available at: www.ibm.com/.../web/library/wa-lucene2/.

[25] Paul de Vrieze, “Improving search engine technology”, Master thesis, 7-Mar-2002.

[26] “Web Crawler or WebRobot or Web Spider Working”, available at: http://codeglobe.blogspot.com/2009/02/web-crawler-or-webrobot-or-web-spider.html

[27] Fabrizio Silvestri, “High Performance Issues in Web Search Engines: Algorithms and Techniques”, Ph.d.Thesis: TD 5/04, available at: http://hpc.isti.cnr.it/_silvestr, May 2004.

[28] M.P.S.Bhatia, Divya Gupta, “Discussion on Web Crawlers of Search Engine”, Proceedings of 2nd National Conference on Challenges & Opportunities in Information Technology (COIT-2008).

[29] Brian Pinkerton, “WebCrawler: Finding What People Want”, Doctor of Philosophy University of Washington, 2000.

List of Publications

Communicated:

- Ravikiran Routhu, Ravinder Kumar, "Enriched in performance of focused web crawler algorithm", International Journal of Computational Intelligence Systems, 2010.