# IMPROVING EFFICIENCY OF WEB-CRAWLER ALGORITHM USING PARAMETRIC VARIATIONS

*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*

**BhaskerReddy KethiReddy**

**(Roll No. 800832001)**

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 the thesis entitled, **"Improving Efficiency Of Web Crawler Algorithm Using Parametric Variations"**, 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 the supervision of Ravinder Kumar and refers other researcher's works which are duly listed in the 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.

**(BhaskerReddy KethiReddy)**

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

15/7/10

**(Mr. Ravinder Kumar)**
Assistant Professor,
Computer Science and Engineering Department,
Thapar University, Patiala.

**Countersigned by**

15/07/10

**(Dr. RAJESH BHATIA)**
Professor & Head,
Computer Science & Engineering Department,
Thapar University,
Patiala.

23/7/10

**(Dr. R.K.SHARMA)**
Dean (Academic Affairs),
Thapar University,
Patiala.

# Certificate

I hereby certify that the work which is being presented in the thesis entitled, **"Improving Efficiency Of Web Crawler Algorithm Using Parametric Variations"**, 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 the supervision of Ravinder Kumar and refers other researcher's works which are duly listed in the 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.

(**BhaskerReddy KethiReddy)**

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**

**(RAJESH BHATIA)**
Professor & Head,
Computer Science & Engineering Department,
Thapar University,
Patiala.

**(R.K.SHARMA)**
Dean (Academic Affairs),
Thapar University,
Patiala.

# Acknowledgement

First and foremost, I would like to express my sincere gratitude to my advisor **Mr. Ravinder Kumar, Assistant Professor,** Computer Science & Engineering Department for his immense help, guidance, stimulating suggestions and encouragement all the time. He always provide a motivating and enthusiastic atmosphere to work with, it was a great pleasure to do this thesis under his supervision.

I am also thankful to **Dr. Rajesh Bhatia,** Head of Department, Computer Science & Engineering Department and **Mrs. Inderveer Channa**, P.G. Coordinator, for the motivation and inspiration that triggered me for the thesis. I would also like to thank all the staff members who were always there at the need of the hour and provided with all the help and facilities, which I required for the completion of the thesis.

Last but not the least, I express my heartfelt thanks to my parents and all of my friends for encouraging me and providing me useful information during my work.

Finally, my special thanks goes to authors whose works I have consulted and quoted in this work.

**BhaskerReddy KethiReddy**

(800832001)

# Abstract

There are billions and billions of Web pages published over the internet via World Wide Web. All of us rely on internet as a source of information. This source of information is available in various forms; Websites, databases, images, sound, videos and many more. A search engine classifies the Search results by keyword matches, link analysis, or other mechanisms perhaps not entirely clear to a front end user. Search engines help us to gather information from their own indexed databases. The Web Crawler of these search engines is expert in crawling various Web pages to gather huge source of information.

In this thesis report explains the basic architecture of crawler based search engine, its working in offline (query independent) and online (query dependent). Also explains architecture of web crawler, its working and various algorithms used by the web crawler for index the web. Calculating the pagerank value to a page by considering web as graph.

A better relevance formula was implemented by combining page rank with the topic similarity measure of the hyper link meta data, this provides better ranking to the relevant documents for the given user query. From the experimental results of metrics which are implemented into C language concludes the implemented formula gives better ranking than other metrics.

# Table of Contents


Certificate i

Acknowledgement ii

Abstract iii

Table of Contents iv

List of Figures vi

List of Tables vii

**Chapter 1:Introduction 1**

**Chapter 2: Search engines and web crawler** 2

2.1 Over View 2

2.2 Architecture and components of crawler- based search engine 5

2.2.1 Examples of crawler based search engines 6

2.2.2 Crawler based search engines carries out two kinds of searching's 6

2.2.3 Web crawler 10

2.2.3.1 Basic Crawling Terminology 13

2.2.3.2 Architecture and Working of Basic Web Crawler 13

2.2.4 Parallel Crawlers 15

**Chapter 3: Crawling Algorithms 21**

3.1 Blind Traversing Approach 21

3.1.1 Breadth First Algorithm 22

3.1.2 Drawbacks of Breadth First Approach 23

3.2 Best –First Heuristic Approach 23

3.2.1 Naive Best - First Algorithm 23

3.2.2 Fish – Search Algorithm 25

3.2.3 Shark Search Algorithm 27

3.3 Relevance Calculation Algorithms 28
3.3.1 The hits algorithm 28
3.3.2 Page Rank Algorithm 30
3.3.3 Cosine Similarity 36
**Chapter 4 Problem Statement 37**
**Chapter 5 Implementation and Results 38**
**Chapter 6 Conclusion and Future Work 46**
6.1 Conclusions 46
6.2 Future work 46
**References 47**
**List of publications 49**

# List of Figures


Figure 2.1: Common architecture of a search engine 5

Figure 2.2: Elements of Search Engine 7

Figure 2.3: Components of a web-crawler 14

Figure 2.4: Sequential flow a crawler 15

Figure 2.5: Structure of parallel crawler 16

Figure 2.6: Pseudo code of basic web crawler 20

Figure 3.1: Breadth First Crawling Approach 22

Figure 3.2: The Pseudo code for Blind search algorithm 23

Figure 3.3: Web-Page With Highest Relevance Is Picked From Any Position 24

Figure 3.4: The Pseudo code for Naive Best First heuristic algorithm 24

Figure 3.5: Fish-search algorithm 26

Figure 3.6: The pseudo code for Shark Search Algorithm 28

Figure 3.7: A simple Network 30

Figure 3.8: PageRank algorithm 33

Figure 5.1: web structure for P1, P2,.....P8 pages 38

Screenshot 5.1: Reading the values for query vector and term document matrix 40

Screenshot 5.2: Normalized term-document matrix A 41

Screenshot 5.3: Normalized term-document matrix q and with ranked pages 42

Screenshot 5.4: Pagerank values for each pages and their ranks 42

Screen shot 5.5: Rank of the document based on cosine+pagerank 43

Screenshot 5.6: Rank of the document based on sum_relevance 44

# List of Tables

Table 5.1: Term document matrix A for the pages 40

Table 5.2: Relevance values of pages for relevance formulas 44

Table 5.3: No of coincidences for relevance formula order with expected order 45

# Chapter 1
# Introduction

The World-Wide-Web (WWW) is a big network where you can get a huge amount of information. The Web is a collection of interconnected documents and other resources, linked by hyperlinks and URLs. The World Wide Web is one of the services accessible via the Internet, along with various others including e-mail, file sharing, online gaming and others described below There are billions and billions of Web pages published over the internet via World Wide Web. All of us rely on internet as a source of information. This source of information is available in various forms; Websites, databases, images, sound, videos and many more. This information is spread on servers all over the world and it is not possible to handle all these data by human. Search engines are one of the most important services to indicate and find all these pages. Without them it wouldn't be possible to get information in a normal and fast way.

To view a Web page on the World Wide Web , the procedure begins either by typing the URL of the page into a Web browser, or by following a hyperlink to that page or resource. The Web browser then initiates a series of communication messages, behind the scenes, in order to fetch and display it. First, the server-name portion of the URL is resolved into an IP address using the global, distributed Internet database known as the domain name system, or DNS. This IP address is necessary to contact and send data packets to the Web server. The browser then requests the resource by sending an HTTP request to the Web server at that particular address. In the case of a typical Web page, the HTML text of the page is requested first and parsed immediately by the Web browser, which will then make additional requests for images and any other files that form a part of the page. All this searching within the Web is performed by the special engines, known as Web Search Engines.

The question is what is going on behind these search engines and why is it possible to get relevant data so fast?

The answer is web crawlers. Web crawlers also named web spiders, web robots or less frequently used names ants, worms or automatic indexers. A web crawler combs

through the internet and copies all pages to index them. They can also be used to harvest e-mail addresses (mostly for spam) or validate HTML code and check links on a page. But they are most in use for search engines to provide up-to-date data to distribute relevant and new search results. The algorithm behind a web robot is quite simple: It starts with a prepared list of links and visits each of these pages. It identifies links on the particular pages and adds them to the list. In a recursive way it visits all links found according to a set of policies.

Most people find what they're looking for on the World Wide Web by using search engines like Yahoo!, Alta Vista, or Google. According to InformationWeek, aside from checking e-mail, searching for information with search engines was the second most popular Internet activity in the early 2000s. Because of this, companies develop and implement strategies to make sure people are able to consistently find their sites during a search. These strategies oftentimes are included in a much broader Web site or Internet marketing plan. Different companies have different objectives, but the main goal is to obtain good placement in search results.

# Chapter 2

# Search Engines and Web Crawler

## 2.1 Over View

**What is a search engine?**

Search engine is a software program that helps in locating information stored on the computer system, typically on the World Wide Web.

**Types of search engines**

In the early 2000s, more than 1,000 different search engines were in existence, although most Web masters focused their efforts on getting good placement in the leading 10. This, however, was easier said than done. InfoWorld explained that the process was more art than science, requiring continuous adjustments and tweaking, along with regularly submitting pages to different engines for good or excellent results. The reason for this is that every search engine works differently. Not only are there different types of search engines—those that use spiders to obtain results, directory-based engines, and link-based engines—but engines within each category are unique. They each have different rules and procedures companies need to follow in order to register their site with the engine. There are three types of search engines:

i. Crawler –based search engines.
ii. Human powered search engines.
iii. Link-based search engines.

**i. Crawler –based search engine**

Many leading search engines use a form of software program called spiders or crawlers to find information on the Internet and store it for search results in giant databases or indexes. 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. Although they usually aren't visible to someone using a Web browser, Meta tags are special codes that provide keywords or Web site descriptions to spiders. Keywords and how they are placed, either within actual Web site content or in Meta tags, are very important to online marketers. The majority of consumers reach e-

commerce sites through search engines, and the right keywords increase the odds a company's site will be included in search results.

Companies need to choose the keywords that describe their sites to spider-based search engines carefully, and continually monitor their effectiveness. Search engines often change their criteria for listing different sites, and keywords that cause a site to be listed first in a search one day may not work at all the next. Companies often monitor search engine results to see what keywords cause top listings in categories that are important to them.

**ii. Human powered search engines**

While some sites use spiders to provide results to searchers, others—like Yahoo!—use human editors. This means that a company cannot rely on technology and keywords to obtain excellent placement, but must provide content the editors will find appealing and valuable to searchers. Some directory-based engines charge a fee for a site to be reviewed for potential listing. In the early 2000s, more leading search engines were relying on human editors in combination with findings obtained with spiders. LookSmart, Lycos, AltaVista, MSN, Excite and AOL Search relied on providers of directory data to make their search results more meaningful.

**iii. Link-based search engines**

One other kind of search engine provides results based on hypertext links between sites. Rather than basing results on keywords or the preferences of human editors, sites are ranked based on the quality and quantity of other Web sites linked to them. In this case, links serve as referrals. The emergence of this kind of search engine called for companies to develop link-building strategies. By finding out which sites are listed in results for a certain product category in a link-based engine, a company could then contact the sites' owners—assuming they aren't competitors—and ask them for a link. This often involves reciprocal linking, where each company agrees to include links to the other's site.

Besides focusing on keywords, providing compelling content and monitoring links, online marketers rely on other ways of getting noticed. In late 2000, some used special software programs or third-party search engine specialists to maximize results

for them. Search engine specialists handle the tedious, never ending tasks of staying current with the requirements of different search engines and tracking a company's placement. This trend was expected to take off in the early 2000s, according to research from IDC and Netbooster, which found that 70 percent of site owners had plans to use a specialist by 2002. Additionally, some companies pay for special or enhanced listings in different search engines.

## 2.2 Architecture and Components of Crawler-Based Search Engine

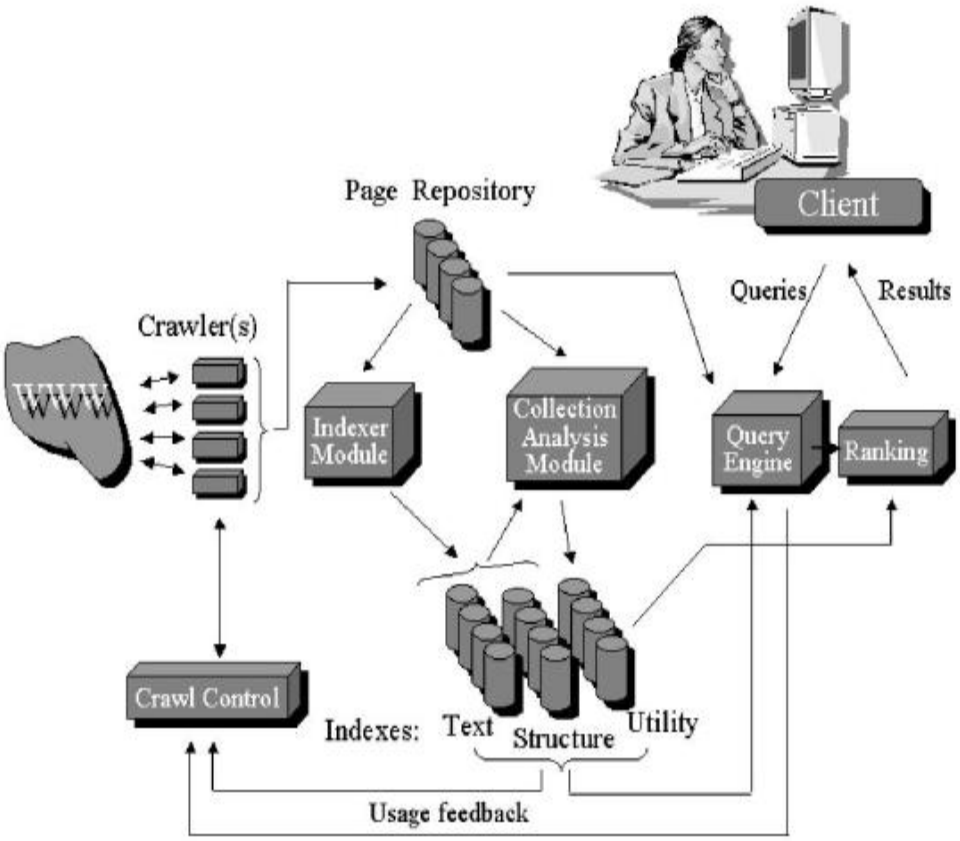



**Figure 2.1: Common architecture of a search engine [1]**

**There are three major elements**:

**i) Crawler**: Also called the spider. The spider visits a web page, reads it, and then follows links to other pages within the site. The spider will return to the site on a regular basis, such as every month or every fifteen days, to look for changes [2].

**ii) Indexer**: Everything the spider finds goes into the second part of the search engine, the index. The index [2] will contain a copy of every web page that the spider finds. If a web page changes, then the index is updated with new information.

**iii) Search engine software**: This is the software program that accepts the user-entered query, interprets it, and sifts through the millions of pages recorded in the index to find matches and ranks them in order of what it believes is most relevant and presents them in a customizable manner to the user.

All crawler-based search engines have the above basic components but differ in the ways these are implemented and tuned.

### 2.2.1 Examples of crawler based search engines:

There are a number of web search engines available in the market. The list of the most important and famous search engines can be listed as below:

- RBSE
- WebCrawler
- World Wide Web Worm
- Google Crawler
- Mercator
- Web Fountain
- Web RACE
- Ubi crawler

### 2.2.2 Crawler Based Search Engines Carries Out Two Kinds Of Searching

- In the indexing mode (query independent)
- In real-time search mode (query dependent)

We can understand the above two processes in the following figure 2.2.

**Indexing mode**

The ultimate goal of any search engine is to build databases of larger indexes. If the WebCrawler index has enough space for 50,000 Web pages, then those Web pages should be more relevant ones. For a Web index [4], one solution is that those Web pages should come from as many different servers as possible. The WebCrawler takes the following approach: it uses a modified breadth-first algorithm to ensure that every server has at least one Web page represented in the index.

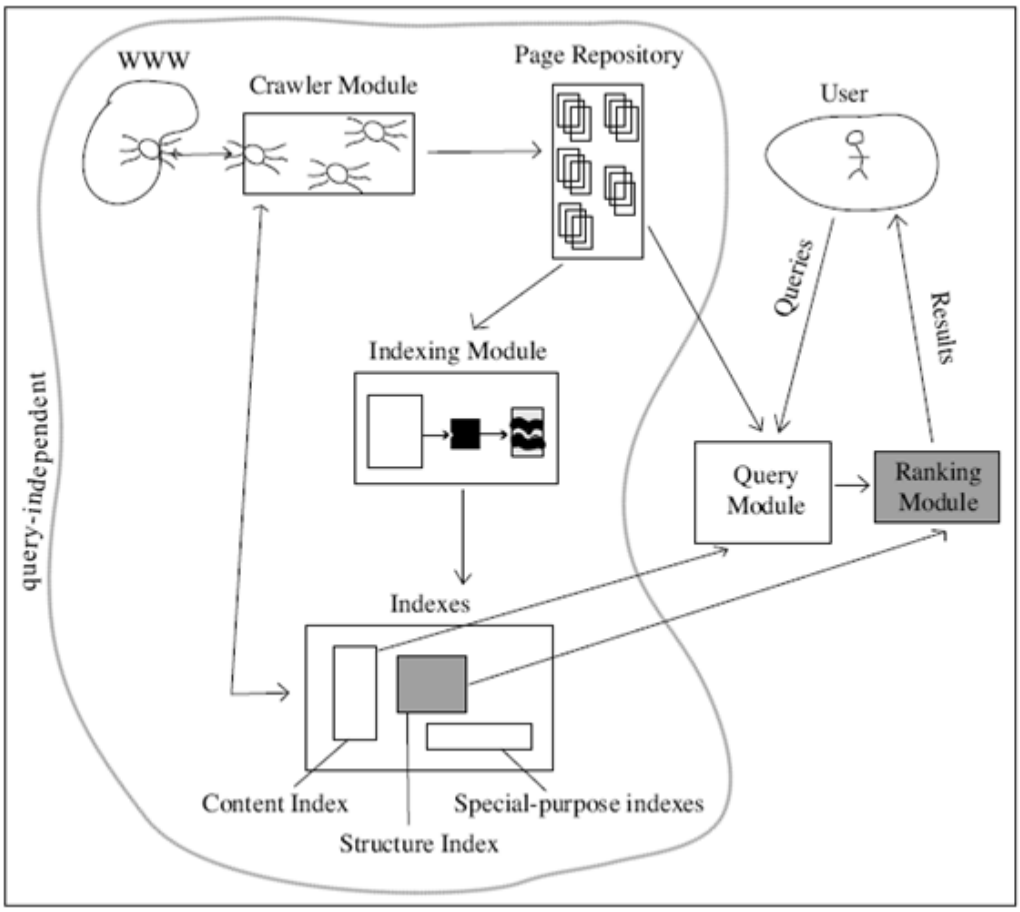



**Figure 2.2: Elements of Search Engine [3].**

This strategy is very effective. The most frequent feedback about the WebCrawler is that it has great coverage and that nearly every server is represented. In detail, a WebCrawler indexing run proceeds as follows: every time a Web page on a new server is found, that server is placed on a list of servers to be visited right away. Before any other Web pages are visited, a Web page on each of the new servers is retrieved and indexed. When all known servers have been visited, indexing proceeds sequentially through a list of all servers until a new one is found, at which point the process repeats. During indexing, the WebCrawler runs either for a certain amount of time, or until it has retrieved some number of Web pages. Normally, the WebCrawler can build an index at the rate of about 1000 Web pages an hour on a 486-based PC running NEXTSTEP.

The four modules above (crawler, page repository, indexers, and indexes) and their corresponding data files exist and operate independent of users and their queries [4]. Spiders are constantly crawl the Web, bringing back new and updated pages to be

indexed and stored. These modules are circled and labelled as query-independent. Unlike the preceding modules, the query module is query-dependent and is initiated when a user enters a query, to which the search engine must respond in real-time.

**Real-time search mode**

The real time search mode is when the WebCrawler has to find the relevant Web pages based on the users query. The key thing in such kind of search mode is to follow the URLs from the Web pages that are similar to what the user wants, and thus these URLs will lead to more relevant Web pages. Such a kind of approach roughly captures the way people navigate the Web: they find a Web page about a topic related to what they are looking for and follow links from there.

In order to find an initial list of similar Web pages, the WebCrawler runs the user's query against its index. A single Web page can also be used as a starting point or the whole category/sub category of the Web directory is used, but using the index is much more efficient. From the list, the most relevant Web pages are noted, and any unexplored links from those Web pages are followed. As new Web pages are retrieved, they are added to the index, and the query is re-run [4]. The results of the query are sorted by relevance, and new Web pages near the top of the list become candidates for further exploration. The process is iterated either until the WebCrawler has found enough similar Web pages to satisfy the user or until a time limit is reached. However, there is a problem with this approach. The WebCrawler blindly follows links from the Web pages, which may end up in following irrelevant path. For e.g. if the WebCrawler is searching for pages about sport news, it may come across Web pages that have sporting accessories. Whenever, people browse through the Web pages they follow the links based on the anchor text, these are the words that describes about other Web pages. People click on these links and traverse in that particular direction.

Ideally, the WebCrawler should choose among several of these links, preferring the one that made the most sense. Although the WebCrawler's reasoning ability is somewhat less than that of a human, it does have a basis for evaluating each link: the similarity of the anchor text to the user's query. To evaluate this similarity, the WebCrawler makes a small full-text index from the anchor text in a document and applies the users query to select the most relevant link. Searching the anchor text in

this way works well, but anchor texts are usually short and full-text indexing does not work as well as it could. More sophisticated full-text tools would help greatly, particularly the application of a thesaurus to expand the anchor text.

The basic idea of following different Web pages from one to another using the links came was first demonstrated to work in the Fish search [5]. The WebCrawler extends that concept to initiate the search using the index, and to follow links in an intelligent order. The query module converts a user's natural language query into a language that the search system can understand (usually numbers), and consults the various indexes in order to answer the query. For example, the query module consults the content index and its inverted file to find which pages use the query terms. These pages are called the relevant pages. Then the query module passes the set of relevant pages to the ranking module [6].

The ranking module takes the set of relevant pages and ranks them according to some criterion. The outcome is an ordered list of WebPages such that the pages near the top of the list are most likely to be what the user desires. The ranking module is perhaps the most important component of the search process because the output of the query module often results in too many (thousands of) relevant pages that the user must sort through. The ordered list filters the less relevant pages to the bottom, making the list of pages more manageable for the user. (In contrast, the similarity measures of traditional information retrieval often do not filter out enough irrelevant pages.) Actually, this ranking which carries valuable, discriminatory power is arrived at by combining two scores, the content score and the popularity score. Many rules are used to give each relevant page a relevancy or content score. For example, many web engines give pages using the query word in the title or description a higher content score than pages using the query word in the body of the page [6]. The popularity score, which is the focus of this book, is determined from an analysis of the Web's hyperlink structure. The content score is combined with the popularity score to determine an overall score for each relevant page [6]. The set of relevant pages resulting from the query module is then presented to the user in order of their overall scores.

### 2.2.3 Web Crawler

Web crawler (also known as a Web spider or Web robot) is a program or automated script which browses the World Wide Web in a methodical and automated manner.

This process is called Web crawling or spidering. Many legitimate sites, in particular search engines, use spidering as a means of providing up-to-date data. Web crawlers are mainly used to create a copy of all the visited pages for later processing by a search engine, which will index the downloaded pages to provide fast searches. Crawlers can also be used for automating maintenance tasks on a Web site, such as checking links or validating HTML codes. Also, crawlers can be used to gather specific types of information from Web pages, such as harvesting e-mail addresses (usually for spam).

A Web crawler is one type of bot, or software agent. In general, it starts with a list of URLs to visit, called the seeds. As the crawler visits these URLs, it identifies all the hyperlinks in the page and adds them to the list of URLs to visit, called the crawl frontier.

**A Survey of Web Crawlers [7]**

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 [8] (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. The crawlers transmitted downloaded pages to a single Store Server 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. Typically, three to four crawler machines were used, so the entire system required between four and eight machines. Research on web crawling continues at Stanford even after Google has been transformed into a commercial effort. The Stanford Web Base 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 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. 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.

**Why do we need a web crawler?**

Following are some reasons to use a web crawler:

- To maintain mirror sites for popular Web sites.
- To test web pages and links for valid syntax and structure.
- To monitor sites to see when their structure or contents change.
- To search for copyright infringements.
- To build a special-purpose index. For example, one that has some understanding of the content stored in multimedia files on the Web.

**How does a web crawler work?**

A typical web crawler starts by parsing a specified web page: noting any hypertext links on that page that point to other web pages. The Crawler then parses those pages for new links, and so on, recursively. A crawler is a software or script or automated program which 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.

This is the basic concept behind implementing web crawler, but implementing this concept is not merely a bunch of programming. The next section describes the difficulties involved in implementing an efficient web crawler[9].

**Difficulties in implementing efficient web crawler**

There are two important characteristics of the Web that generate a scenario in which Web crawling is very difficult:

1. Large volume of Web pages.
2. Rate of change on web pages.

A large volume of web page implies that web crawler can only download a fraction of the web pages and hence it is very essential that web crawler should be intelligent enough to prioritize download.

Another problem with today's dynamic world is that web pages on the internet change very frequently, as a result, by the time the crawler is downloading the last page from a site, the page may change or a new page has been placed/updated to the site.

**Solutions - Right strategies**

The difficulties in implementing efficient web crawler clearly state that bandwidth for conducting crawls is neither infinite nor free. So, it is becoming essential to crawl the web in not only a scalable, but efficient way, if some reasonable amount of quality or freshness of web pages is to be maintained. This ensures that a crawler must carefully choose at each step which pages to visit next. Thus the implementer of a web crawler must define its behaviour. Defining the behaviour of a Web crawler is the outcome of a combination of below mentioned strategies:

- Selecting the better algorithm to decide which page to download
- Strategizing how to re-visit pages to check for updates
- Strategizing how to avoid overloading websites

**Selecting the right algorithm**

Given the current size of the web, it is essential that the crawler program should crawl on a fraction of the web. Even large search engines in today's dynamic world crawls

fraction of web pages from web. But, a crawler should observe that the fraction of pages crawled must be most relevant pages, and not just random pages.

While selecting the search algorithm for the web crawler an implementer should keep in mind that algorithm must make sure that web pages are chosen depending upon their importance. The importance of a web page lies in its popularity in terms of links or visits, or even it's URL.

#### 2.2.3.1 Basic Crawling Terminology

**Seed Page**

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.

**Frontier (Processing Queue)**

The crawling method starts with a given URL (seed), extracting links from it and adding them to an un-visited list of URLs. This list of un-visited links or URLs is known as, "**Frontier".** Each time, a URL is picked from the frontier by the Crawler Scheduler**.** This frontier is implemented by using Queue, Priority Queue Data structures. The maintenance of the Frontier is also a major functionality of any Crawler.

**Parser**

Once a page has been fetched, we need to parse its content to extract Information that will feed and possibly guide the future path of the crawler. Parsing May Imply simple hyperlink/URL extraction or it may involve the more complex process of tidying up the HTML content in order to analyze the HTML tag tree. The job of any parser is to parse the fetched web page to extract list of new URLs from it and return the new un-visited URLs to the Frontier.

#### 2.2.3.2 Architecture and Working of Basic Web Crawler

A typical web crawler starts by parsing a specified web page: noting any hypertext links on that page that point to other web pages. The Crawler then parses those pages for new links, periodically return to the sites to check for any information that has

changed. The frequency with which this happens is determined by the administrators of the search engine

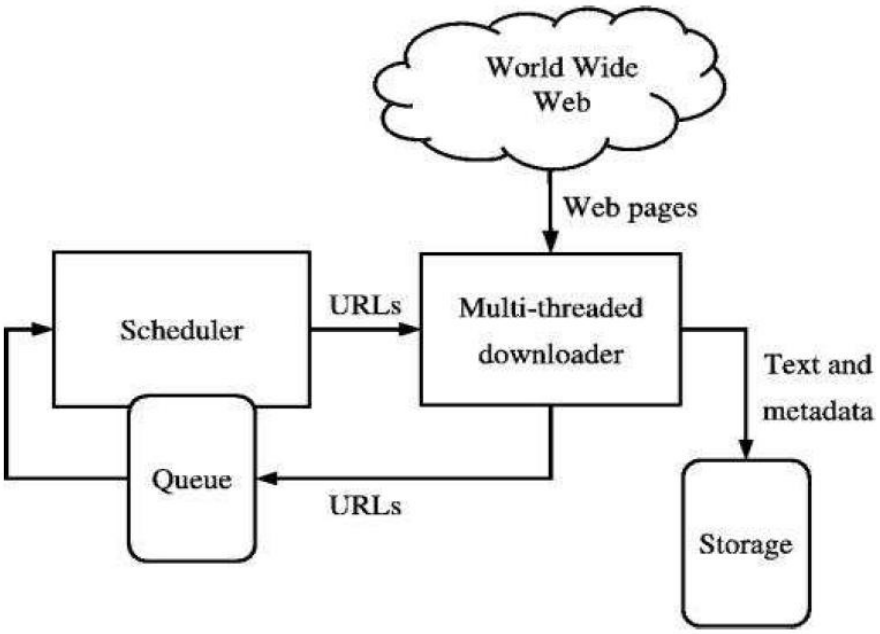



**Figure 2.3: Components of a web-crawler [10].**

Common web crawler implements method composed from following steps:

- Acquire URL of processed web document from processing queue
- Download web document
- Parse document's content to extract set of URL links to other resources and
- Update processing queue
- Store web document for further processing
- The basic working of a web-crawler can be discussed as follows:
- Select a starting seed URL or URLs
- Add it to the frontier
- Now pick the URL from the frontier
- Fetch the web-page corresponding to that URL
- Parse that web-page to find new URL links
- Add all the newly found URLs into the frontier
- Go to **step 2** and repeat while the frontier is not empty

Thus a crawler will recursively keep on adding newer URLs to the database repository of the search engine. So we can see that the main function of a crawler is to

add new links into the frontier and to select a new URL from the frontier for further processing after each recursive step [11]. The working of the crawlers can also be shown in the form of a flow chart. Such crawlers are called sequential crawlers because they follow a sequential approach.

**In simple form, the flow chart of a web crawler can be stated as below:**

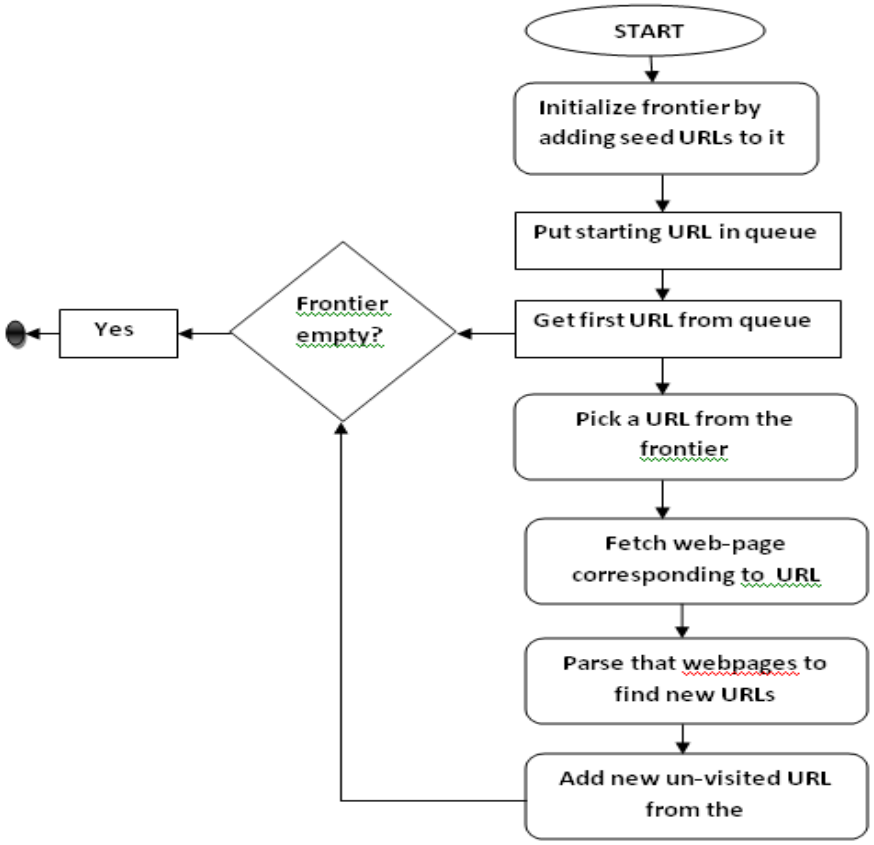



**Figure 2.4: Sequential flow a crawler**

### 2.2.4 Parallel Crawlers

As the size of the Web grows, it becomes more difficult to retrieve the whole or a significant portion of the Web using a single sequential crawler. Therefore, many search engines often run multiple processes in parallel to perform the above task, so that download rate is maximized. We refer to this type of crawler as a **parallel crawler**. Parallel crawlers work simultaneously to grab pages from the World Wide Web and add them to the central repository of the search engine. The parallel crawling architecture is shown in the figure. Each crawler is having its own database of collected pages and own queue of un-visited URLs. Once the crawling procedure

finishes, the collected pages of every crawler are added to the central repository of the search engine.

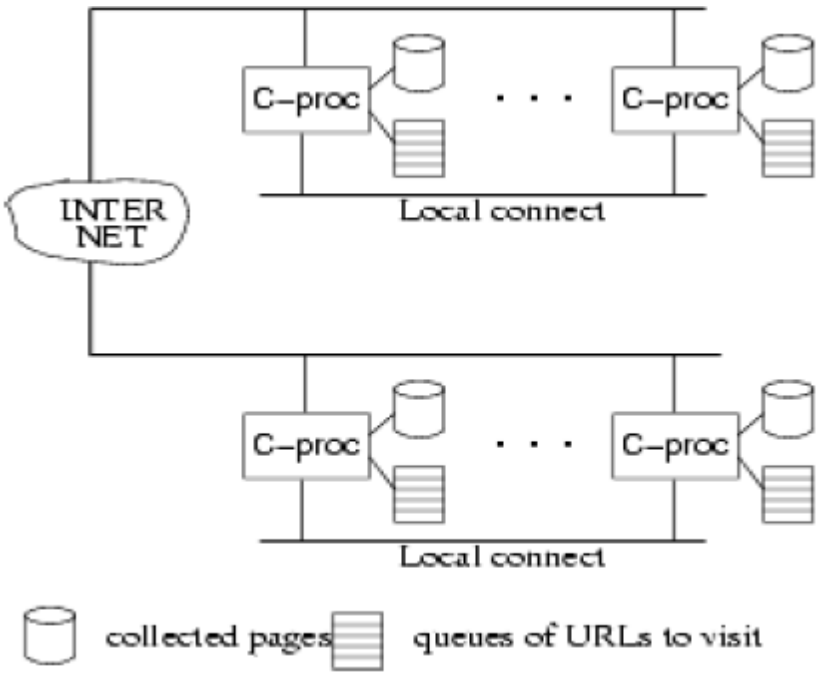



**Figure 2.5: Structure of parallel crawler [12]**

Parallel crawling architecture no doubt increases the efficiency of any search engine but there are certain problems or issues related with the usage of parallel crawlers.

**Examples of Web crawlers [10]**

The following is a list of published crawler architectures for general-purpose crawlers (excluding focused Web crawlers), with a brief description that includes the names given to the different components and outstanding features:

**RBSE** was the first published Web crawler. It was based on two programs: the first program, "spider" maintains a queue in a relational database, and the second program "mite", is a modified www ASCII browser that downloads the pages from the Web.

**WebCrawler** was used to build the first publicly-available full-text index of a sub-set of the Web. It was based on lib-WWW to download pages, and another program to parse and order URLs for breadth-first exploration of the Web graph. It also included a real-time crawler that followed links based on the similarity of the anchor text with the provided query.

**World Wide Web Worm** 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.

**Internet Archive Crawler** is a crawler designed with the purpose of archiving periodic snapshots of a large portion of the Web. It uses several processes in a distributed fashion, and a fixed number of Web sites are assigned to each process. The inter-process exchange of URLs is carried in batch with a long time interval between exchanges, as this is a costly process. The Internet Archive Crawler also has to deal with the problem of changing DNS records, so it keeps an historical archive of the hostname to IP mappings.

**Web SPHINX** is composed of a Java class library that implements multi-threaded Web page retrieval and HTML parsing, and a graphical user interface to set the starting URLs, to extract the downloaded data and to implement a basic text-based search engine.

**Google Crawler** is described in some detail, but the reference is only about an early version of its architecture, 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 an 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.

**CobWeb** Uses a central "scheduler" and a series of distributed "collectors". The collectors parse the downloaded Web pages and send the discovered URLs to the scheduler, which in turns assign them to the collectors. The scheduler enforces a breadth-first search order with a politeness policy to avoid overloading Web servers. The crawler is written in Perl.

**Mercator** is a modular Web crawler written in Java. Its modularity arises from the usage of interchangeable "protocol modules" and "processing modules". Protocols modules are related to how to acquire the Web pages (e.g.: by HTTP), and processing modules are related to how to process Web pages. The standard processing module

just parses the pages and extracts new URLs, but other processing modules can be used to index the text of the pages, or to gather statistics from the Web.

**WebFountain** is a distributed, modular crawler similar to Mercator but written in C++. It features a "controller" machine that coordinates a series of "ant" machines. After repeatedly downloading pages, a change rate is inferred for each page and a non-linear programming method must be used to solve the equation system for maximizing freshness. The authors recommend using this crawling order in the early stages of the crawl, and then switching to a uniform crawling order, in which all pages being visited with the same frequency.

**PolyBot** is a distributed crawler written in C++ and Python, which is composed of a "crawl manager", one or more "downloaders" and one or more "DNS resolvers". Collected URLs are added to a queue on disk, and processed later to search for seen URLs in batch mode. The politeness policy considers both third and second level domains (e.g.: www.example.com and www2.example.com are third level domains) because third level domains are usually hosted by the same Web server.

**WebRACE** 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 changes, 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.

**Ubicrawler** is a distributed crawler written in Java, and it has no central process. It is composed of a number of identical "agents"; and the assignment function is calculated using consistent hashing of the host names. There is zero overlap, meaning that no page is crawled twice, unless a crawling agent crashes (then, another agent must re-crawl the pages from the failing agent). The crawler is designed to achieve high scalability and to be tolerant to failures.

**FAST Crawler** is the crawler used by the FAST search engine, and a general description of its architecture is available. It is a distributed architecture in which each

machine holds a “document scheduler” that maintains a queue of documents to be downloaded by a “document processor” that stores them in a local storage subsystem. Each crawler communicates with the other crawlers via a “distributor” module that exchanges hyperlink information.

**Pseudo code of basic web crawler**

Here's a pseudo code summary of the algorithm that can be used to implement a web crawler:

```
Add the URL to the empty list of URLs to search.

While not empty (the list of URLs to search)
{
  Take the first URL in from the list of URLs
  Mark this URL as already searched URL.

  If the URL protocol is not HTTP then
        break;
        go back to while

  If robots.txt file exist on site then
        If file includes Disallow statement then
              break;
              go back to while

  Open the URL

  If the opened URL is not HTML file then
        Break;
        Go back to while

  Iterate the HTML file

  While the html text contains another link {

        If robots.txt file exist on URL/site then
              If file includes Disallow statement then
                    break;
                    go back to while
```

```
                If the opened URL is HTML file then
                        If the URL isn't marked as searched then
                                Mark this URL as already searched URL.

                        Else if type of file is user requested
                        Add to list of files found.

        }
    }
```

**Figure 2.6: Pseudo code of basic web crawler.**

# Chapter 3
# Crawling Algorithms

Crawlers exploit the Web's hyperlinked structure to retrieve new pages by traversing links from previously retrieved ones. As pages are fetched, their outward links may be added to a list of unvisited pages, which is referred to as the crawl frontier. A key challenge during the progress of a topical crawl is to identify the next most appropriate link to follow from the frontier. The algorithm to select the next link for traversal is necessarily tied to the goals of the crawler. A crawler that aims to index the Web as comprehensively as possible will make different kinds of decisions than one aiming to collect pages from university Web sites or one looking for pages about movie reviews. For the first crawl order may not be important, the second may consider the syntax of the URL to limit retrieved pages to the .edu domain, while the third may use similarity with some source page to guide link selection. Even crawlers serving the same goal may adopt different crawl strategies.

Two major approaches used for crawling are:

- Blind Traversing approach
- Best – First Heuristic approach

## 3.1 Blind Traversing Approach

In this approach, we simply start with a seed URL and apply the crawling process as stated earlier. It is called blind because for selecting next URL from frontier, no criteria are applied. Crawling links are selected in the order in which they are encountered in the frontier (in serial order) One algorithm widely common to implement Blind traversing approach is – Breadth First Algorithm. It uses FIFO QUEUE data structure to implement the frontier; it is very simple and basic crawling algorithm. Since this approach traverses the graphical structure of WWW breadth – wise, Queue data structure is used to implement the Frontier. Algorithm that comes under Blind Crawling approach is Breadth First Algorithm.

### 3.1.1 Breadth First Algorithm

A Breadth-First crawler is the simplest strategy for crawling. This algorithm was explored in 1994 in the WebCrawler as well as in more recent research [12]. It uses the frontier as a FIFO queue, crawling links in the order in which they are encountered. The problem with this algorithm is that when the frontier is full, the crawler can add only one link from a crawled page. 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.

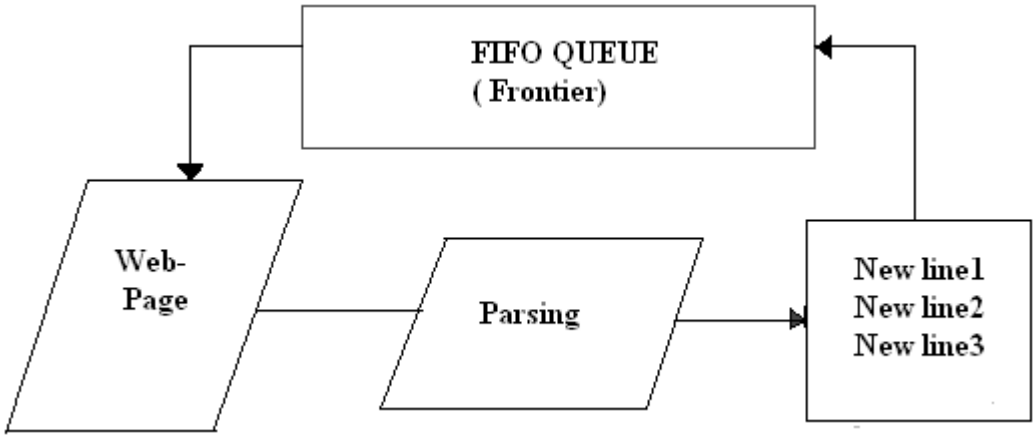



**Figure 3.1: Breadth First Crawling Approach**

That is why, also called, Blind Search Algorithm. Its performance is used to provide a lower bound for any of the more sophisticated algorithms.

```
Breadth-First ( starting_urls) {
                    For each link (starting_urls) {
                    Enqueue( frontier, link);
                     }
                While ( visited < MAX_PAGES ) {
                     Link := deque_link( frontier);
                          doc := fetch(link);
                     enqueue(frontier, extract_links (doc))
                     If (# frontier > MAX_BUFFER){
```

```
                dequeue_last_links(frontier);
                }
            }
        }
```

**Figure 3.2: The Pseudo code for Blind search algorithm.**

### 3.1.2 Drawbacks 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 our repository. Problem with Blind Breadth First algorithm is that it traverses URLs in sequential order as these were inserted into the Frontier. 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.

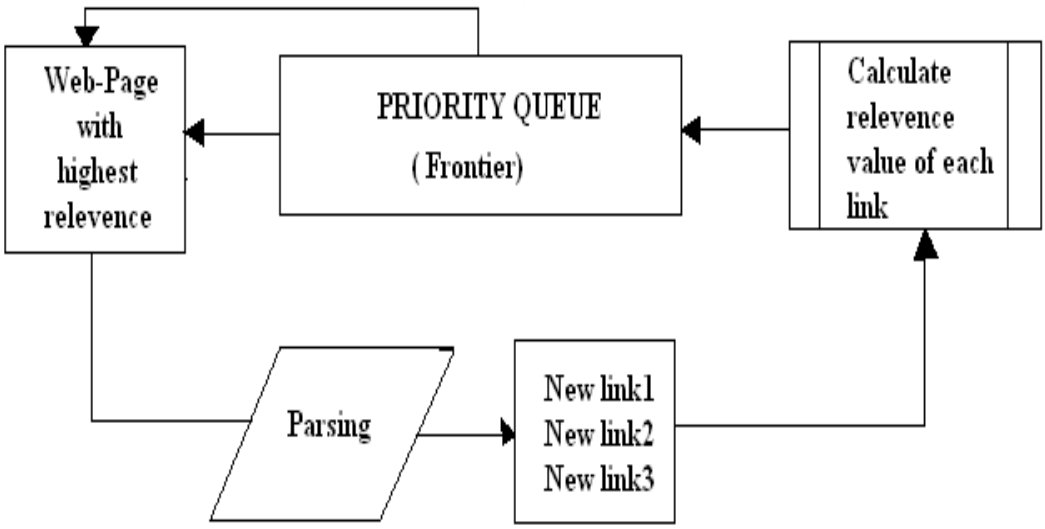
## 3.2 Best - First Heuristic Approach

To overcome the problems of blind traverse approach, a heuristic approach called Best-First crawling approach have been studied by Cho and Hersovici. 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) Following Web Crawling Algorithms use Heuristic Approach.

### 3.2.1 Naive Best - First Algorithm

One Best First Approach uses a relevancy Function rel ( ) to compute the lexical similarity between the desired key-words and each page & associate that value with

corresponding links in the frontier. After each iteration, the link with the highest rel ( ) function value is picked from the frontier.




.

**Figure 3.3: Web-Page With Highest Relevance Is Picked From Any Position From Frontier For Processing.**

```
BFS( topic, starting_urls) {
For each link (starting_urls) {
Enqueue( frontier, link,1);
}
While ( visited < MAX_PAGES ) {
Link := deque_top_link( frontier);
doc := fetch(link);
score := sim(topic, doc)
enqueue(frontier, extract_links (doc),score)
If (# frontier > MAX_BUFFER){
dequeue_bottom_links(frontier);
}
} }
```

**Figure 3.4: The Pseudo code for Naive Best First heuristic algorithm.**

That is, the best available link is traversed every time which is not possible in Breadth First Approach. Since any link with highest relevancy value can be picked from the Frontier, most of the Best first algorithms use Priority Queue as data structure. The working of Heuristic Crawling Algorithms is illustrated in figure3.4.
As clear from figure 3.3, web-page with highest relevance is picked from any position from Frontier for processing.

### 3.2.2 Fish – Search Algorithm

Web search services typically use a previously built index that is actually stored on the search service server(s). This approach is pretty efficient for searching large parts of the Web, but it is basically static. The actual search is performed on the server on all the data stored in the index. Results are not guaranteed to be valid at the time the query is issued (note that many search sites may take up to one month for refreshing their index on the full Web). In contrast, dynamic search actually fetches the data at the time the query is issued. While it does not scale up, dynamic search guarantees valid results, and is preferable to static search for discovering information in small and dynamic sub-Webs. One of the first dynamic search heuristics was the "fish search" [12], 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, I 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 fish search algorithm** can be stated as follows:

**Get** as input parameters, the initial node, depth (D) and the time limit, and

a search query

**Set** the depth of the initial node as depth=D, and Insert into an empty list

**While** the list is not empty, and the time limit is not reached

**Pop** the first node from the URL list and make it the current_node

**Compute** the relevance of the current_node

**If** depth>0

(1 )**If** current_node is relevant

**Then For** each child_node of current_node

**Add** to the front of the list;

**Set** potential_score(child_node)=1;

**Else For** each child_node of current_node

**Add** to the list right after the last node;

**Set** potential_score(child_node)=0;

(2) **For** each child_node of current_node

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

**If** current_node is relevant

**Then Set** depth(child_node)=D

**Else** depth(child node)=depth(current_node)- 1

**If** child_node already exists in the URL list

**Then Compute** the maximum between the existing depths in the list to the newly

Computed depth;

**Replace** the existing depth in the list by that maximum;

**End While**

**Figure 3.5: Fish-search algorithm.**

The fish-search algorithm, while being attractive because of the simplicity of its paradigm, and its dynamic nature, presents some limitations. First, 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. Clearly, the main issue that needs to be addressed is a finer grained scoring capability. This is problematic because it is difficult to assign a more precise potential score to documents which have not yet been fetched.

### 3.2.3 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 [11] 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.

```
Shark (topic, starting_urls) {
        for each link (starting_urls) {
        set_depth (link, d)
            enqueue (frontier, link);
}
While ( visited < MAX_PAGES ) {
        Link := deque_top_link(frontier);
        doc := fetch(link);
        doc_score := sim(topic, doc);
        if (depth(link) > 0) {
            for each outlink (extract_links(doc)) {
            score = (1-r) * neighbourhood_score(outlink)
                        +  r * inherited_score(outlink);
        if (doc_score > 0) {
```

```
                set_depth (outlink, d);
                    }else{
            Set_depth (outlink, depth(link) - 1);
            }
            enqueue(frontier, outlink, score);
            }
            If (# frontier > MAX_BUFFER){
            dequeue_bottom_link(frontier);
            }
        }
    }
```

**Figure 3.6: The pseudo code for Shark Search Algorithm.**

The Shark Search Algorithm improved working of Fish Search algorithm in a lot of manner. The child inherits the discounted value of ancestors and Shark Search also keeps in consideration the Anchor text of a web page while assigning it any relevancy value.

## 3.3 Relevance Calculation Algorithms

### 3.3.1 The hits algorithm

HITS [13], a link analysis algorithm developed by Jon Kleinberg from Cornell University during his postdoctoral studies at IBM Almaden, aimed to focus this long, unruly query list. The HITS algorithm is based on a pattern Kleinberg noticed among Web pages. Some pages serve as hubs or portal pages, i.e., pages with many outlink. Other pages are authorities on topics because they have many inlinks [14]. Kleinberg noticed that good hubs seemed to point to good authorities and good authorities were pointed to by good hubs. So he decided to give each page i both an hub score hi and an authority score $a_i$. In fact, for every page i he defined the hub score at iteration k,$h_i^{(k)}$ and the authority score, $a_i^{(k)}$ , as

$$a_i^{(k)} = \sum_{j:e_{ji} \in E} h_j^{(k-1)} \ and \ h_i^{(k)} = \sum_{j:e_{ji} \in E} a_j^{(k)} \ for \ k = 1,2,3, \ldots$$

Where, $e_{ij}$ represents a hyperlink from page i to page *j* and *E* is the set of hyperlinks. To compute the scores for a page, he started with uniform scores for all pages, i.e., $h_i^{(1)} = 1/n$ and $a_i^{(1)} = 1/n$ where *n* is the number of pages in a so-called neighbourhood set for the query list. The neighbourhood set consists of all pages in the query list plus all pages pointing to or from the query pages. Depending on the query, the neighbourhood set could contain just a hundred pages or a hundred thousand pages. (The neighbourhood set allows latent semantic associations to be made.) The hub and authority scores are iteratively refined until convergence to stationary values. Using linear algebra we can replace the summation equations with matrix equations. Let a and h be column vectors holding the authority and hub scores. Let L be the adjacency matrix for the neighbourhood set. That is, $L_{ij} = 1$ if page i links to page j, and 0, otherwise. These definitions show that

$$a^{(k)} = L^T h^{(k-1)} \text{ and } h^{(k)} = La^{(k)}.$$

Using some algebra, we have

$$a^{(k)} = L^T La^{(k-1)}$$

$$h^{(k)} = LL^T h^{(k-1)}$$

These equations make it clear that Kleinberg's algorithm is really the power method applied to the positive semi-definite matrices $L^T L$ and $LL^T$ . $L^T L$ is called the hub matrix and $LL^T$ is the authority matrix. Thus, HITS amounts to solving the eigenvector problems $L^T La = \lambda_1 a$ and $LL^T h = \lambda_1 h$, where $\lambda_1$ is the largest eigenvalue of $LL^T$ (and $LL^T$ ), and a and h are corresponding eigenvectors. While this is the basic linear algebra required by the HITS method, there are many more issues to be considered. For example, important issues include convergence, existence, uniqueness, and numerical computation of these scores [25, 26, and 27]. Several modifications to HITS have been suggested, each bringing various advantages and disadvantages.

### 3.3.2 Page Rank Algorithm

**Motivation for page rank**

When using search engines to look for information on the Internet, we often find that, while much of what we find is useful, a great deal of it is also irrelevant to our queries, making it difficult to separate the useful information from the useless. Furthermore, search results also sometimes include irrelevant commercial web pages masquerading as relevant sources of information, further complicating matters. Thus, search engines are charged with a major task: to decide quantitatively what content is relevant and authoritative, thus making it easier to find useful information. Google, one of the most popular search engines at the time of writing, uses a well-known algorithm, PageRank, whose job is to assign authority rankings to every page in Google's World Wide Web index. This document discusses PageRank.

Page Rank was proposed by Brin and Page [14] as a possible model of user surfing behaviour. The PageRank of a page represents the probability that a random surfer (one who follows links randomly from page to page) will be on that page at any given time. A page's score depends recursively upon the scores of the pages that point to it. Source pages distribute their PageRank across all over their outlinks. Consider the illustration in figure 3.7 of a simple network of web pages, which we will use to illustrate PageRank.

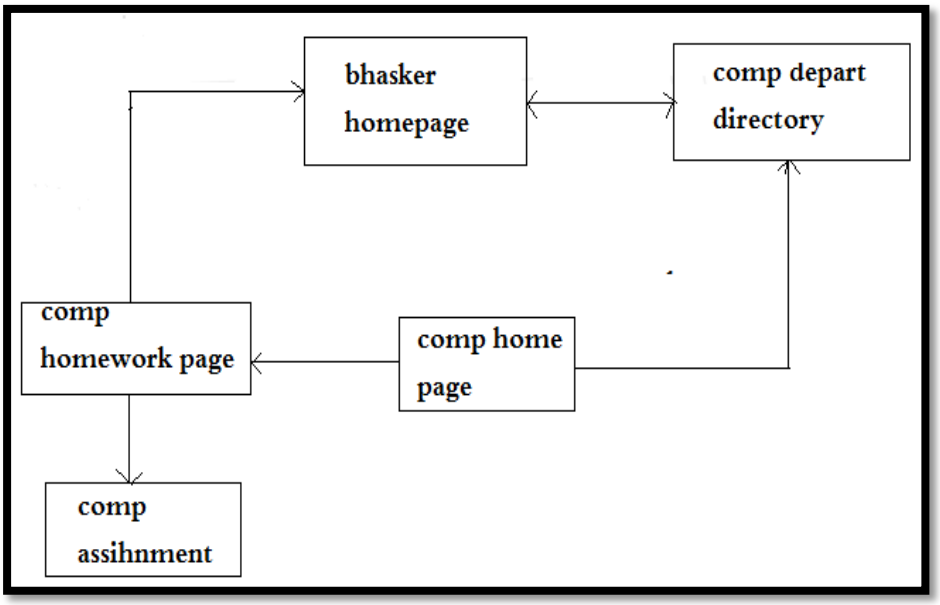



**Figure 3.7: A simple Network.**

Mathematically, we can think of this network as a graph, where each page is a vertex, and a link from one page to another is a graph edge [15]. Furthermore, we can represent this network using an n×n adjacency matrix A, where Aij = 1 if there is a link from vertex i to vertex j, and 0 otherwise [15]. In the language of PageRank, vertices are nodes (web pages), the edges from a node are forward links, and the edges into a node are backlinks [17]. Thus, the 5×5 adjacency matrix for the graph in figure 3.7 is

$$A=\begin{bmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad (1)$$

Notice that there are no forward links from node five. Pages with no outlinks are called dangling nodes, and require special treatment. Furthermore, notice that the diagonal entries are all zero. We assume that links from a page to itself are ignored when constructing an adjacency matrix.

**Informal description**

An informal description of PageRank is given by an intuitive definition of how one might measure the importance of a web page on the Internet. A page p's PageRank is based upon how many other pages link to p. Specifically, the PageRank of p is the sum of the PageRanks of each page qi that links to p divided by the number of pages to which qi links. This intuitive definition makes sense: if p is linked to only by pages with low PageRanks, it is probably not authoritative. Further, if p is linked to by a page v with a high PageRank, but v links to many other pages, p should not receive the full weight of v's PageRank.

**Definition**

**Define PageRank as follows**:

• Let u be a web page.

• Let Fu be the set of forward links from u

• Let Bu be the set of backlinks into u

• Let Nu = |Fu| be the number of forward links from u

• Let c be a normalization factor so that "the total rank of all web pages is constant" [17]

• Let E(u) be "some vector over the Web pages that corresponds to a source of rank" [17]

Then, the PageRank of page u is given by [11]

$$R'(u) = c\sum_{v \in B_u} \frac{R'(v)}{N_v} + cE(u), \qquad (2)$$

Which, is iterated until the values have converged sufficiently.

**Random surfer**

PageRank also has a second definition based upon the model of a random web surfer navigating the Internet. In short, the model states that PageRank models the behaviour of someone who "keeps clicking on successive links at random" [17]. However, occasionally the surfer "gets bored and jumps to a random page chosen based on the distribution in E." [17].

As originally proposed PageRank was intended to be used in combination with content based criteria to rank retrieved sets of documents. This is in fact how PageRank is used in the Google search engine. More recently PageRank has been used to guide crawlers and to assess page quality.

The algorithm of page rank is given in figure 3.8

```
                    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);
        }
    }
```

**Figure 3.8: PageRank algorithm.**

**(i)The Google Matrix**

Although PageRank can be described using equation 2, the summation method is neither the most interesting nor the most illustrative of the algorithm's properties. The preferable method is to rewrite the sum as a matrix multiplication. If we let $\pi^T$ be the 1×n PageRank row vector, and H the n×n row-normalized adjacency matrix, then we can describe the PageRank vector at the kth iteration as

$$\pi^{(K+1)T} = \pi^{(K)T}\mathrm{H}, \qquad (3)$$

So that successive iterations $\pi^{(K+1)T}$ converge to the PageRank vector $\pi^T$

**(ii) Row-Normality and Stochasticity**

Although we have mentioned it, we have not yet formed the matrix H. Thus, the first step is to let $H_i = \frac{A_i}{\sum_{k=1}^{n} Aik}$ ,

So that each row Ai of A is divided by its sum (the reasons for which will become clear later). Using A from equation 1,

We have,

$$H = \begin{bmatrix} 0 & 1/2 & 0 & 1/2 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1/2 & 0 & 0 & 0 & 0 \\ 1/3 & 1/3 & 1/3 & 0 & 1/2 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \qquad (4)$$

Using the matrix H is insufficient for the PageRank algorithm, however, because the iteration using H alone might not converge properly — "it can cycle or the limit may be dependent on the starting vector". Part of the explanation for this is that the matrix H is not yet necessarily stochastic.

A matrix is stochastic when it is a "matrix of transition probabilities for a Markov chain,"1 with the property that "all elements are non-negative and all its row sums are unity" (one). Thus, to ensure that H is stochastic, we must ensure that every row sums to one. But clearly, the sum of the last row of the H in equation 4 is zero. Therefore, we define the stochastic S as

$$S = H + \frac{ae^T}{n}, \quad (5)$$

**(iii) Irreducibility**

However, we are still not completely done, because there is no guarantee that S has a unique stationary distribution vector (i.e., there might not be a single "correct" PageRank vector). For us to guarantee that there is a single stationary distribution vector _T to which we can converge, we must ensure that S is irreducible as well as stochastic. A matrix is irreducible if and only if its graph is strongly connected (defined shortly), so we define the irreducible row-stochastic matrix G as

$$G = \alpha S + (1-\alpha)E, \quad (6)$$

Where, $0 \leq \alpha \leq 1$ and $E = \mathbf{e}\frac{e^T}{n}$. G is the Google matrix, and we can use it to define

$$\pi^{(K+1)T} = \pi^{(K)T}\mathbf{G}, \quad \mathbf{(7)}$$

**(iv) The Power Method**

When dealing with data sets as large as Google uses (more than eight billion web pages [10]), it is unrealistic to form a matrix G and find its dominant left eigenvector. It is more efficient to compute the PageRank vector using the power method, where we iterate using the sparse matrix H by rewriting equation (7). In other words, from [4],

$$\boldsymbol{\pi}^{(K)T} = \boldsymbol{\pi}^{(K-1)T}\mathbf{G}$$

$$= \pi^{(K-1)T}(\alpha S + (1-\alpha)e\frac{e^T}{n})$$

$$= \alpha\pi^{(K-1)T}S + (1-\alpha)\,\pi^{(K-1)T}\,\frac{\boldsymbol{e^T}}{\boldsymbol{n}}$$

$$= \alpha\pi^{(K-1)T}S + (1-\alpha)\,\mathbf{e}\frac{\boldsymbol{e^T}}{\boldsymbol{n}}$$

$$= \alpha\pi^{(K-1)T}(H + \frac{ae^T}{n}) + (1-\alpha)\,\mathbf{e}\frac{\boldsymbol{e^T}}{\boldsymbol{n}}$$

$$= \alpha\pi^{(K-1)T}H + (\,\alpha\pi^{(K-1)T}a + (1-\alpha))\frac{\boldsymbol{e^T}}{\boldsymbol{n}}.$$

**Strengths and Weaknesses of PageRank**

We have already mentioned one weakness of PageRank-the topic drift problem due to the importance of determining an accurate relevancy score. Much work, thought, and heuristics must be applied by Google engineers to determine the relevancy score, otherwise, no matter how good PageRank is the ranked list returned to the user is of little value if the pages are off-topic This invites the question, Why does *importance* serve as such a good proxy to relevance? Or does it? By emphasizing the importance of documents, are lesser-known, obscure, yet highly relevant documents being missed? Bharat and Mihaila succinctly state this weakness of PageRank in [18]. "Since PageRank is query independent, it cannot by itself distinguish between pages that are authoritative in general and pages that are authoritative on the query topic." Some of these questions may be unanswerable due to the proprietary nature of Google, but they are still worthy of consideration.

On the other hand, the use of importance, rather than relevance, is the key to Google's success and the source of its strength. By measuring importance, query dependence, the main weakness of HITS, becomes a nonissue. Instead, the PageRank measure of importance is a query-independent measure [24]. At query time, only a quick look up into inverted file storage is required to determine the relevancy set, which is then sorted by the pre computed PageRanks.

Another, strength of PageRank is its virtual imperviousness to spamming. As mentioned during the HITS discussion, it is very hard for a webpage owner to add inlinks into his page from other important pages. Chien [19] have proven that if the owner succeeds in doing this, the PageRank is guaranteed to increase. However, this

increase will likely be inconsequential since PageRank is a global measure [20, 21]. In contrast, HITS' authority and hub scores are derived from a local neighborhood graph and slight increases in the number of inlinks or outlinks will have a greater relative impact. Thus, in summary, webpage owners have very little ability- and are not very likely-to affect their PageRank scores. Nevertheless, there have been some papers describing ways to both influence PageRank and recognize spam attempts [22, 23].

### 3.3.3 Cosine Similarity

Now we use a cosine similarity measure to calculate the relevance of the page on a particular topic.

$$\text{Relevence}\,(t, p) = \frac{\sum k \varepsilon\,(t \cap p) wkt\, wkp}{\sqrt{\sum k \varepsilon\, t\ (wkt)^2 \ \sum\ k \varepsilon\, t\,(wkp)^2}}$$

Here, *t* is the topic specific weight table, p is the web page under investigation, $w_{kt}$ and $w_{kp}$ is the weight of keyword *k* in the weight table and in the web page respectively. The range of Relevance (t, p) lies between 0 and 1, and the relevancy increases as this value increases [16]. If the relevance score of a page is greater than relevancy limit specified by the user, then this page is added to database as a topic specific page.

A number of methods for crawling are currently used by the search engines. The two approaches of searching and crawling are - Basic Blind Search approach which uses Breadth First Algorithm. Any good crawling algorithm should address certain issues. The other more powerful heuristic Approach is currently used by most of the search engines. The algorithms using this approach are – Best First, Fish Search and Shark Search Algorithms.

# Chapter 4

# Problem Statement

The PageRank algorithm is often used in ranking web pages, and it is also used in URL ordering for focused crawler. The main advantage of Page Rank is its ability to fight spam. A page is important, if the pages pointing to it are more. Since it is not easy for Web owner to add in links into his/her page from other important pages, it is not easy to influence Page Rank. Page Rank is a global measure. However, it assigns each outlink the same weight and it is query independent. The Page Rank could not distinguish between pages that are authoritative in general and pages that are authoritative on the query topic.

Similarity measure is concentrate on text to calculate relevance of the documents for ranking. However Similarity measure gives an advantage to keyword spammers to increase their home pages weight by repeating keywords many times. So, combination of these two (PageRank and Similarity measure) will eliminate the drawbacks in ranking the documents for a given user query.

## **Objectivies** :

- Exploring the relevance formulas in ranking the documents
- Identifying the problems in relevance formulas to distinguish between pages that are authoritative in general and pages on the query topic
- For solving the problem implemented formula by combining page rank with the topic similarity measure.

# Chapter5

# Implementation and Results

This chapter describes calculation of the cosine formula, page rank for a given query by considering some pages related to the query. Observing the results of maximum, minimum of cosine , page rank formulas, and also finding the values for page rank with average of cosine values of inlinks.

QUERY: LATENT SEMANTIC INDEXING ALGORITHM.

P1. Analysis of semantic indexing and latent structure.

P2. Advance of semantic analysis.

P3.Semantic indexing tutorials and books.

P4. Latent heat and steam and latent heat of ice.

P5.Latent semantic analysis matlab.

P6. Algorithm for semantic indexing .

P7. Algorithm of polynomial semantic indexing.

P8.Learning latent semantic indexing algorithm.

The above web pages can be represented in graph as

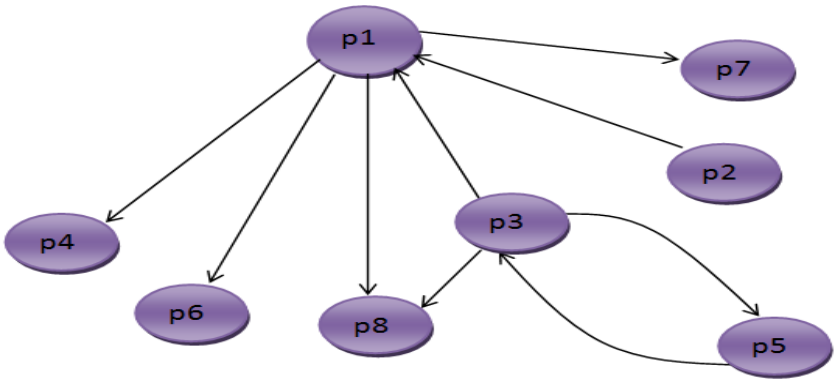



**Figure 5.1: web structure for P1, P2,.....P8 pages.**

For a given user query we have the collection of eight pages and we are getting the order which is most relevant for given query is : $P_8 > P_6 > P_7 > P_3 > P_5 > P_1 > P_4 > P_2$.

Now, calculating the relevance values for each document using the formulas

Cosine formula

Pagerank

Cosine formula + pagerank

Max( cosine , pagerank)

Min( cosine , pagerank)

Pagerank +(average of cosine values of inlinks)

**Cosine values for given query and its corresponding pages :**

Step 1 : compute A and q ,where A is term document matrix obtained by TF-IDF method and q is query vector.

Step 2 : Normalize A , q.

Step 3 : compute $q^T$ * A .

**Step1 :**

To calculate query vector, comparing each term with query vector

If the term is presented in query, then the value is 1,otherwise 0.

For example, the term “Analysis” is not present in query, so 1st value is 0.

**Query vector q**

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Query | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |

Calculate TF-IDF for each term corresponding to each page,

For example, take the word “Analysis” corresponding to page p1

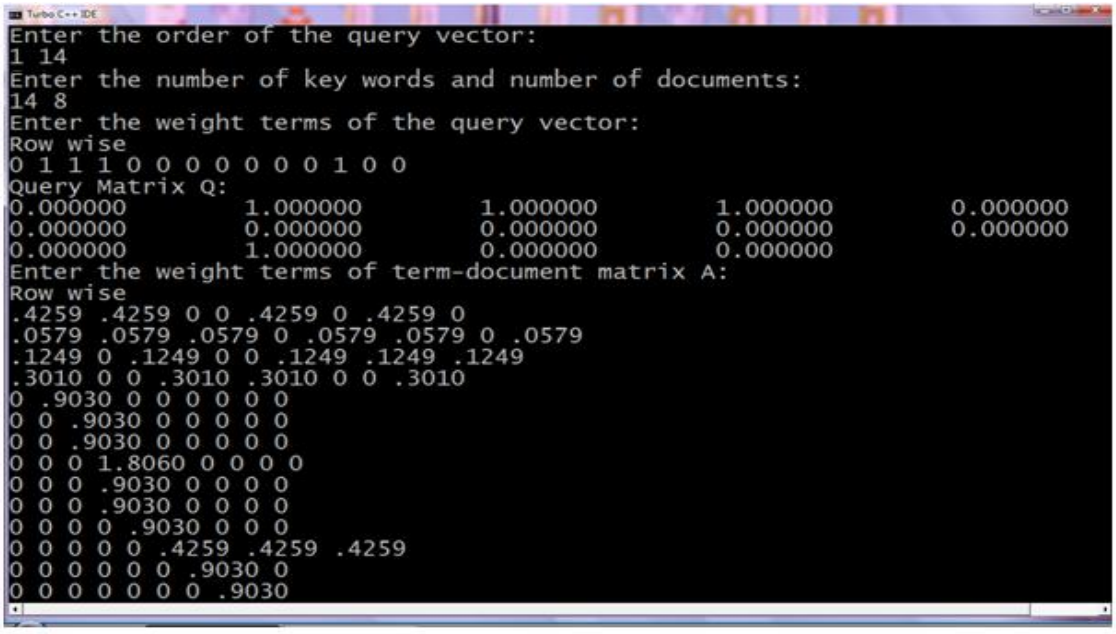
TF-IDF(Analysis) = TF * log(n/m) ,where TF is the frequency of term “Analysis” in page p1, “ n” is the collection size and “m” is the number of pages containing term “Analysis”.

For our problem, TF-IDF(Analysis) = 1 * log(8/3) = 0.4259.

**Table 5.1: Term document matrix A for the pages.**

| Terms | P1 | P2 | P3 | P4 | P5 | P6 | P7 | P8 |
|---|---|---|---|---|---|---|---|---|
| Analysis | .4259 | .4259 | 0 | 0 | .4259 | 0 | .4259 | 0 |
| Semantic | .0579 | .0579 | .0579 | 0 | .0579 | .0579 | 0 | .0579 |
| Indexing | .1249 | 0 | .1249 | 0 | 0 | .1249 | .1249 | .1249 |
| Latent | .3010 | 0 | 0 | .3010 | .3010 | 0 | 0 | .3010 |
| Advance | 0 | .9030 | 0 | 0 | 0 | 0 | 0 | 0 |
| Tutorial | 0 | 0 | .9030 | 0 | 0 | 0 | 0 | 0 |
| Book | 0 | 0 | .9030 | 0 | 0 | 0 | 0 | 0 |
| Heat | 0 | 0 | 0 | 1.8060 | 0 | 0 | 0 | 0 |
| Steam | 0 | 0 | 0 | .9030 | 0 | 0 | 0 | 0 |
| Ice | 0 | 0 | 0 | .9030 | 0 | 0 | 0 | 0 |
| Matlab | 0 | 0 | 0 | 0 | .9030 | 0 | 0 | 0 |
| Algorithm | 0 | 0 | 0 | 0 | 0 | .4259 | .4259 | .4259 |
| Polynomial | 0 | 0 | 0 | 0 | 0 | 0 | .9030 | 0 |
| Learning | 0 | 0 | 0 | 0 | 0 | 0 | 0 | .9030 |

Converting step 1,step 2 and step 3 into code and it takes Table 5.1, Table 5.2 values as input :




**Screenshot 5.1: Reading the values for query vector and term document matrix**.

**Step 2 and 3 :**

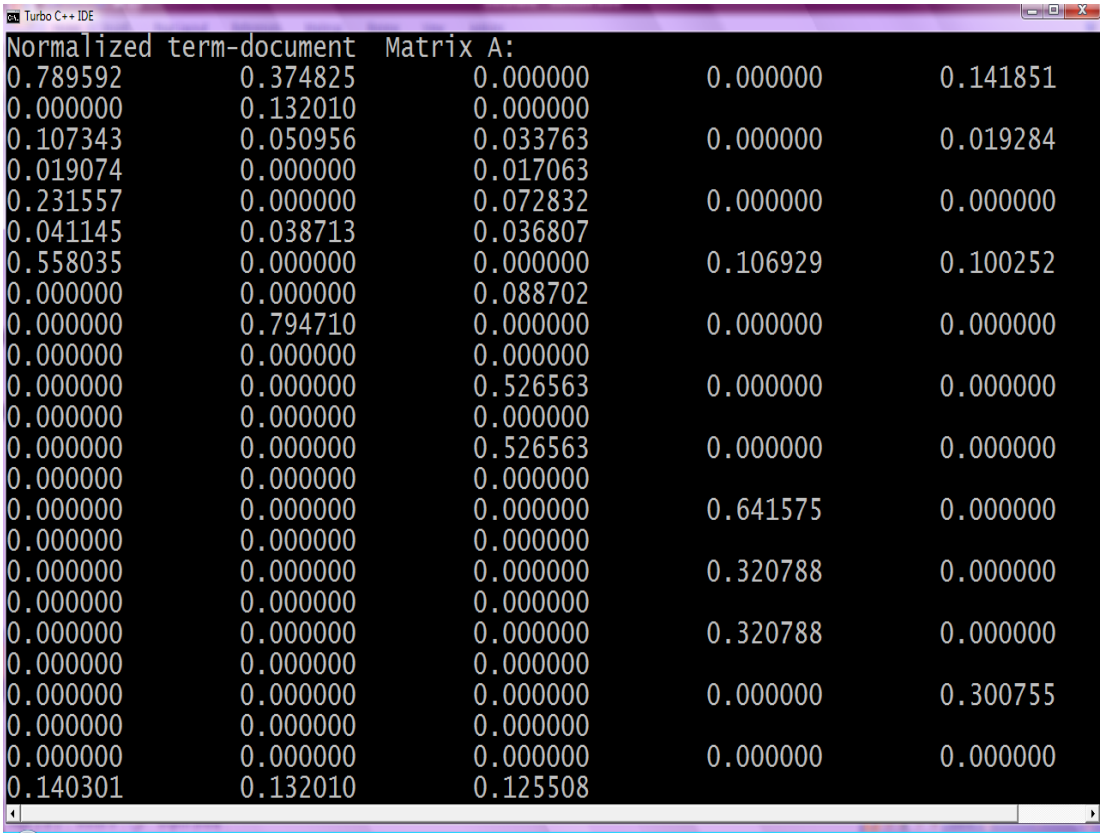
**(i) calculation of the normalized matrix for A, q:** Finding the square root of sum of the squares of the column values of the term document matrix A and divide each column value by this square root value, similarly calculate the normalize matrix for q.

**(ii) Find the $q^T$ * A:**

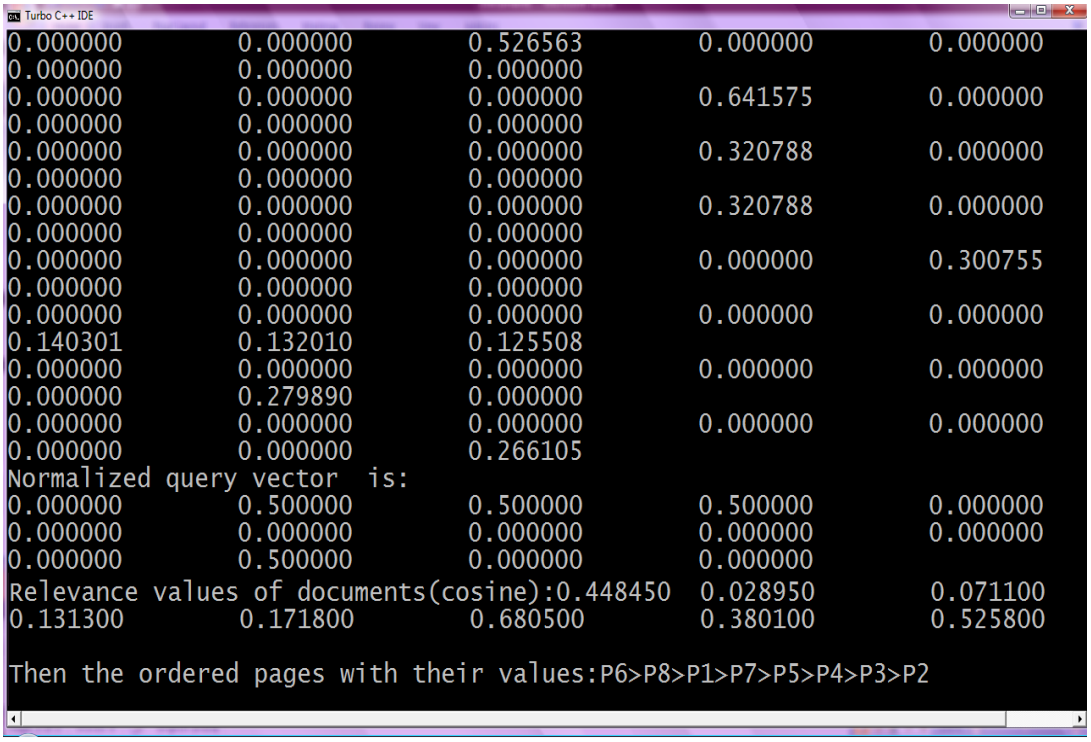
$q^T$ : Transpose matrix of q i.e convert the column values into row values.

Calculating the product of the transposed matrix $q^T$ with A. The result matrix gives the relevance values of each page.

**(iii) Rank the pages based on relevance values calculated in (ii).**

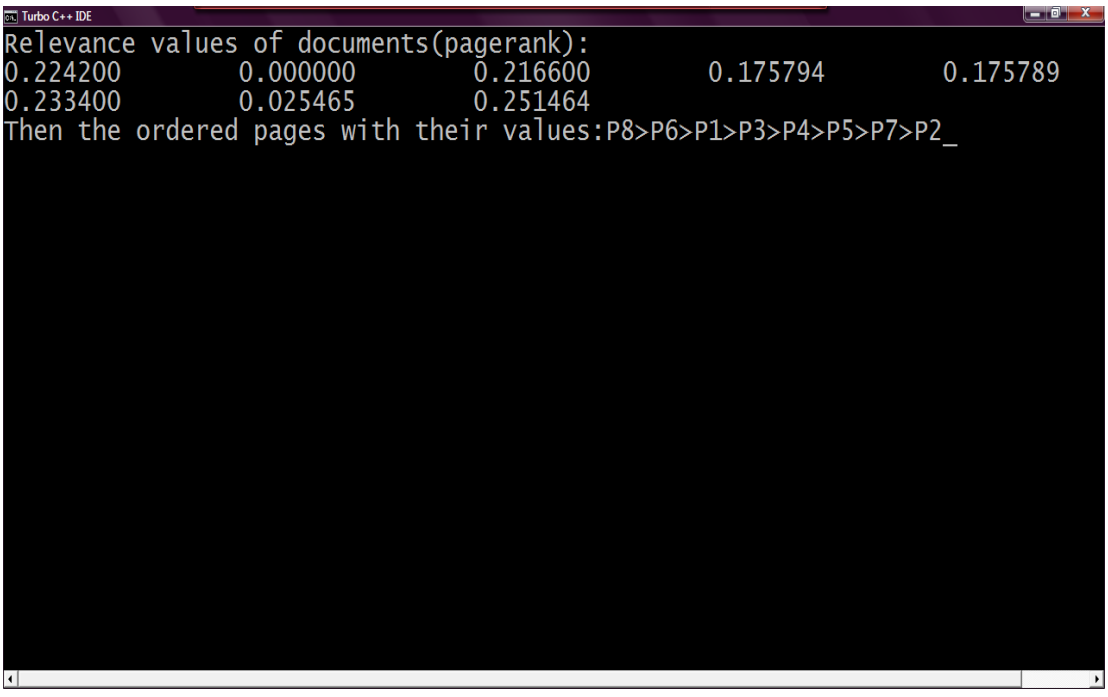



**Screenshot 5.2: Normalized term-document matrix A.**



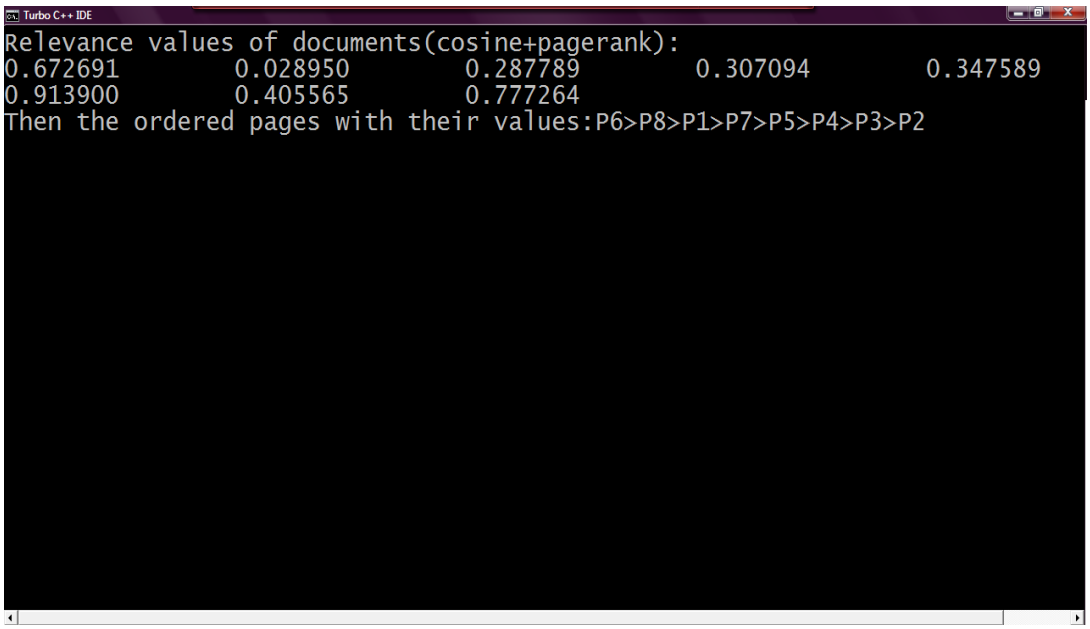

**Screenshot 5.3: Normalized term-document matrix q and with ranked pages.**

Resultant ranked pages are not much more relevant order to the user query. The relevant order improved by calculation of pagerank. Pagerank considers the number of inlinks for a particular page then it assigns the values.




**Screen shot 5.4: Pagerank values for each pages and their ranks.**

Pagerank considers only the number of in links to page but it does not consider the text. The rank of the relevance values of the pages can be improved by considering both pagerank and cosine similarity. By adding cosine values with value of the pagerank it improves relevance order as shown in below screen shot




**Screen shot** 5.5**: Rank of the document based on cosine+pagerank.**

**Improvement in relevance order of pages for a given query:**

**sum_relevance:**

With the use of pagerank, cosine values of inlinks to the page, defining the sum_relevance function as

sum_relevance = pagerank + (average of cosine values of its parents).

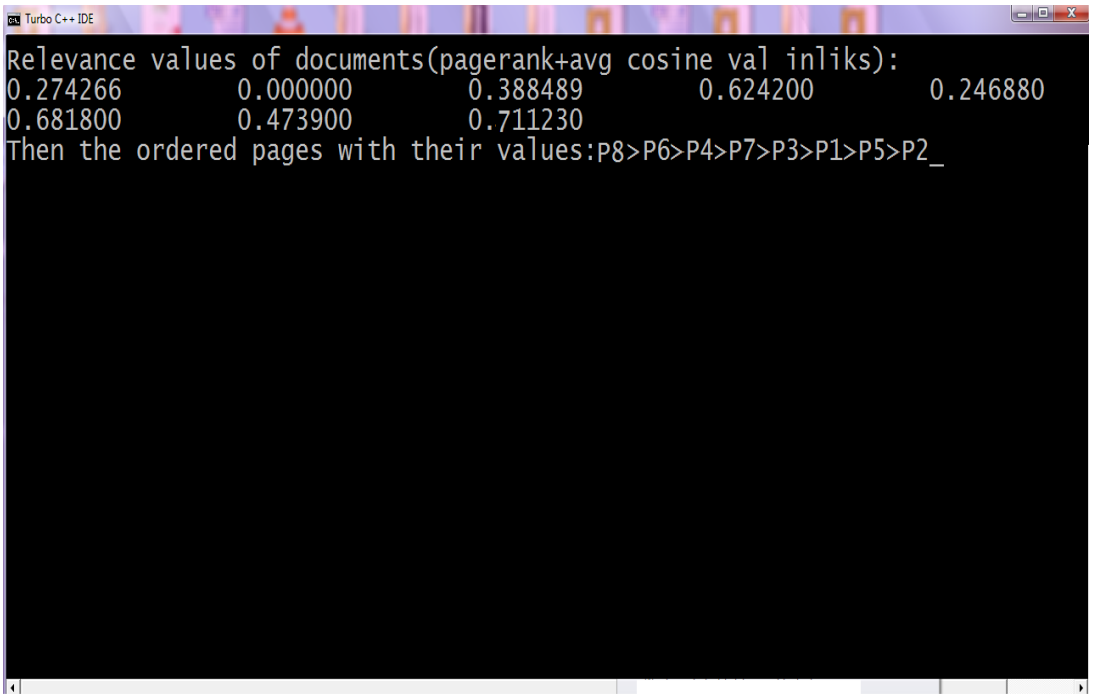
Step1: Calculate the pagerank .

Step2: Calculate the cosine values of its parents.

Step3: Find the average value of values of Step2.

Step4: Find the sum of the Step1 and Step3.

This formula uses pagerank and cosine values of inlinks to that page for getting accurate relevancy of that document for the given user query. From these values order the documents such that most relevant pages displayed in top for a given user query .




**Screenshot 5.6: Rank of the documents based on sum_relevance.**

**From the experimental results:**

**Table 5.2: Relevance values of pages for relevance formulas.**

| Pages | Value of cosine formula | Value of page rank | Value of page rank +cosine | Max(cosine, page rank) | Min(cosine, page rank) | sum_relevance |
|---|---|---|---|---|---|---|
| P1 | .44845 | .224241 | .672691 | .44845 | .224241 | .274266 |
| P2 | .02895 | 0 | .02895 | .02895 | 0 | 0 |
| P3 | .0711 | .216689 | .287789 | .216689 | .0711 | .388489 |
| P4 | .1313 | .175794 | .307094 | .175794 | .1313 | .624244 |
| P5 | .1718 | .175789 | .347589 | .175789 | .1718 | .246889 |
| P6 | .6805 | .2334 | .9139 | .6805 | .2334 | .68185 |
| P7 | .3801 | .025465 | .405565 | .3801 | .025465 | .473915 |
| P8 | .5258 | .251464 | .777264 | .5258 | .251464 | .711239 |

After reordering the pages with scores for each relevance formula ,then we get

Cosine formula: $P_6 > P_8 > P_1 > P_7 > P_5 > P_4 > P_3 > P_2$.

Pagerank : $P_8 > P_6 > P_1 > P_3 > P_5 > P_7 > P_4 > P_2$.

Cosine + pagerank : $P_6 > P_8 > P_1 > P_7 > P_5 > P_4 > P_3 > P_2$.

Max (cosine ,pagerank) : $P_6 > P_8 > P_1 > P_7 > P_3 > P_4 > P_5 > P_2$ .

Min (cosine , pagerank) : $P_7 > P_8 > P_6 > P_1 > P_5 > P_4 > P_3 > P_2$.

Pagerank + (average of cosine values of inlinks): $P_8 > P_6 > P_4 > P_7 > P_3 > P_1 > P_5 > P_2$.

**Coincidence:**

For a given user query, the expected order which is most relevant is : $P_8 > P_6 > P_7 > P_3 > P_5 > P_1 > P_4 > P_2$ .

The order given by Cosine formula : $P_6 > P_8 > P_1 > P_7 > P_5 > P_4 > P_3 > P_2$.

Comparing the corresponding pages from left to right of the above two orders, the number of coincidences is 2.

**Table 5.3: No of coincidences for relevance formula order with expected order.**

| **Relevance formula** | **No of coincidences** |
|---|---|
| Cosine formula | 2 |
| Pagerank | 3 |
| Cosine formula + pagerank | 2 |
| Max( cosine , pagerank) | 1 |
| Min( cosine , pagerank) | 1 |
| sum_relevance | 4 |

Above table concludes that the order of the pages of given query is more relevant when considering the formula sum_relevance.

# Chapter 6
# Conclusions and Future Work

## 6.1 Conclusions

In this thesis basic architecture of crawler based search engine, it's working in offline (query independent) and online (query dependent) were explained. The relevance formulas and various algorithms explained briefly in ranking the documents which are used by the web crawler. These metrics compute the relevance value for each document on the basis of text present in the document, number inlinks and outlinks. A better relevance formula was implemented by combining page rank with the topic similarity measure of the hyper link meta data, this provides better ranking to the relevant documents for the given user query. From the experimental results shown in chapter 5 concludes the implemented formula gives better ranking than other metrics.

## 6.2 Future Work

Different metrics are using to retrieve relevant documents for the given user query. These metrics are independent to each other. So, in future to get more relevant documents by considering the combination of metrics based on the text, inlinks, outlinks.

# References


[1] Philipp Astm, Michael Kapfenberger, and Stefan Hauswiesner, "Crawler Approaches And Technology", November 2008.

[2] Lovekeshkumar desai, "A distributed approach to crawl domain specific hidden web", georgia state university, 2007.

[3] Amy N.Langville and Carl D.meyer, "Google's page rank and beyond", Princeton University Press, June 2006.

[4] Milan c pandya, "A domain based approach to crawl the hidden web", Georgia State University, 2006.

[5] DeBra and R. D. J. *Post*, "Information Retrieval in the World-Wide Web: Making Client-based searching feasible", Proceedings of the second international WWW conference, Mosaic and the Web, 1994.

[6] Brian Pinkerton, "web crawler:Finding what people want", University of Washington,2000.

[7] Marc Najork and Allan Heydon, "High-Performance Web Crawling", SRC Research Report 173, published by COMPAQ systems research centre on September 26, 2001.

[8] Sergey Brin and Lawrence Page, "The anatomy of a large-scale hyper textual Web search engine", In Proceedings of the Seventh International World Wide Web Conference, pages 107–117, April 1998.

[9] Shalin Shah, "Implementing An Efficient Web Crawler", 2003.

[10] Carlos Castillo, "Effective Web Crawling", University of Chile, November 2004.

[11] Sandeep Sharma and Ravinder Kumar, **"**Web-Crawlers and Recent Crawling Approaches", in International Conference on Challenges and Development on IT - ICCDIT-2008 held in PCTE, Ludhiana (Punjab) on May 30th, 2008.

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

[13] Jon Kleinberg, "Authoritative sources in a hyperlinked environment" Journal of the ACM, 46, 1999.

[14] Sergey Brin and Lawrence Page, "The Anatomy of a Large-Scale Hyper textual Web Search Engine", Computer Science Department, Stanford University, Stanford, CAA vailable at- << http://www.his.se/upload/51108/google.pdf >>.

[15] Eric W. Weisstein, Graph, From Math World — A Wolfram Web Resource, December 11th, 2005. *http://mathworld.wolfram.com/Graph.html*.

[16] F. Yuan, C. Yin and Y. Zhang "An application of Improved PageRank in focused Crawler" IEEE 2007.

[17] Lawrence Page, Sergey Brin, Rajeev Motwani, and TerryWinograd, The PageRank Citation Ranking: Bringing Order to the Web (1998). *http://dbpubs.stanford.edu:8090/pub/1999-66.*

[18] K. Bharat and G. A. Mihaila, "When experts agree: Using non-affiliated experts to rank popular topics", ACM Trans, Inform Systems, 20 (2002), pp. 47–58.

[19] S. Chien, C. Dwork, R. Kumar, and D. Sivakumar, "Towards exploiting link evolution", in Proceedings of the Workshop on Algorithms and Models for the Web Graph, 2001.

[20] M. Bianchini, M. Gori, and F. Scarselli, "PageRank: A circuital analysis", in Proceedings of the Eleventh International World Wide Web Conference, ACM, New York, 2002.

[21] A. Y. Ng, A. X. Zheng, and M. I. Jordan, "Link analysis, eigenvectors and stability", in Proceedings of the Seventh International Joint Conference on Artificial Intelligence, 2001.

[22] M. Bianchini, M. Gori, and F. Scarselli, "Inside PageRank", ACM Trans, Internet Tech, 2005.

[23] A. C. Tsoi, G. Morini, F. Scarselli, and M. Hagenbuchner, "Adaptive ranking of webpages", in Proceedings of the Twelfth International World Wide Web Conference, ACM, New York, 2003.

[24] Amy N. Langville and Carl D. Meyer, "A Survey of Eigenvector Methods forWeb Information Retrieval", National Science Foundation under grants CCR-0318575, http://www.siam.org/journals/sirev/47-1/42478.html,2005.

[25] Chris H. Q. Ding, Hongyuan Zha, Xiaofeng He, Parry Husbands, and Horst D. Simon Link analysis: hubs andauthorities on the World Wide Web. SIAM Review, 46(2):256–268, 2004.

[26] Ayman Farahat, Thomas Lofaro, Joel C. Miller, Gregory Rae, and Lesley A. Ward. Existence and uniqueness of ranking vectors for linear link analysis. In ACM SIGIR Conference, September 2001.

[27] Amy N. Langville and Carl D. Meyer. A survey of eigenvector methods of web information retrieval. The SIAMReview, 2003. Accepted in December 2003.

# List of Publications

**Communicated:**

- BhaskerReddy KethiReddy and Ravinder Kumar, "Improving Efficiency Of Web Crawler Algorithm Using Parametric Variations", The Eighth International Conference On Simulated Evolution And Learning (SEAL), 2010.