CHAPTER 3
THE INFORMATION RETRIEVAL TOOLS:
ISSUES AND CHALLENGES

3.1 Introduction
A search system is an information retrieval system designed to minimize the
time required to find the information over the Web of hyperlinked document. It
provides a user interface that enables the user to specify the criteria about an
item of interest and searches the same from locally maintained databases. The
criteria are referred as a search query. In the case of text search systems, the
search query is typically expressed as a set of words that identify the desired
concept that one or more documents may contain. Some text search systems
require users to enter two or three words separated by white spaces for the search
of required information contained in text documents, picture file, audio and
video files. The list of items that meets the criteria specified by the query is
generally stored in sorted or ranked order. Ranking reduces the time required to
find the desired information. In any search system crawling is the first stage that
downloads the Web documents, which are indexed by the indexer for later use
by searching module with a feedback from other stages. This module could also
provide on-demand crawling services for search systems, if required. This
chapter discusses the issues and challenges involved in the design of various
types of search services like search engine and meta search engine etc.

3.2 Web Directories
Web directories represent the Web pages in a hierarchical form. The hierarchical
representation of Web pages provides the information to the end user in the topic
and subtopic form. This hierarchy can grow up to many levels. The human
editors are responsible for review, classification and presentation of topics that
are added to a directory. The Web directory system requires enough interaction
and attention of the end user for searching of information. The directory
structure is easy to use and user does not require any advanced knowledge for exactly what he is looking for in order to find the desired information. The user just has to select the category for the topic in which he is interested and to move down through the hierarchy by selecting sub categories. This narrows the end user search at each level. This process is repeated several times until he is presented with a list of hyperlinks that contains the information of his interest.

3.2.1 Issues and Challenges with Web Directories

While traversing the directory structure downwards, the user is presented with more special topics. But, when going upward, he is heading back to more general topics. Web directories are useful for exploring the topics or related areas but not at a very detailed level. Now a day, Web directories like “the100list” are taking the help of search engines for their enhancement. The various issues and challenges with the parallel crawlers are as follows:

(i) **Indexing of limited number of Web pages:** As the human editor is responsible for maintaining the repository of the Web directories, so limited number of Web pages is maintained. It may also be possible that for some time no results can be displayed. This is quite common for some categories of directories.

(ii) **Expensive and impossible to maintain:** The Web pages in Web directories are maintained by a group of experts. It is an impossible and a time consuming process to maintain the pages available on the Web. So, the resources required to organize and maintain these Web pages are also intensive (Shen et al. 2004).

(iii) **Updating of database is very rare:** As everything of Web directories is done manually, so the Web pages available in Web directories are rarely removed or re-evaluated. That’s why; the interest of the end user for searching the information on such a dynamic WWW by using Web directories is gradually decreased.
3.3 Search Engine

The second approach for organizing and locating the information on the Web is search engine. It is a computer program that performs the following tasks:

(i) It provides the user interface for searching the information on the Web. The user can submit his query on this interface. This query must consist of words or phrases describing the specific information of user interest.

(ii) It searches its database corresponding to the given query.

(iii) It returns all clickable URL that are found matching to the given query.

(iv) This list provides better matched URL link on the top of the returned URL list. These returned URLs may consist of links to other Web pages, textual data, images, audio, video etc.

(v) It allows the user to use advanced search options for searching the information (if provided by the search engine).

In the earliest generation, the search engines downloaded the Web pages and maintained their repository. This repository was maintained by indexing all the Web pages. For searching the user given query search engine used the query matching technique in which the content of the Web page was analyzed corresponding to the frequency and location of occurrence of words (Gurrin and Smeaton 2004). For example – if the query keywords given on search engine interface are matched in the title of the Web page then this Web page would lead to the resultant list of URLs. This URL of the Web page can be followed by the URLs of the other Web pages where the search query terms occurred several times. The query matching techniques did not include the terms like “to”, “is”, “the” etc. in frequency of terms. Moreover, while searching the query terms in the Web page, the following parameters were also not considered by the search engine.
• **Syntax:** The position of the query terms in the Web document was not considered while returning the search results to the user.

• **Synonyms:** While searching the given query, the synonyms of the query terms were not considered.

• **Redundancy:** The words with multiple meanings were not identified.

According to the study in (Wang and DeWitt 2004), in earlier generations, there was no hit for more than 30% of the user query. This problem was introduced, because, in earlier generations only Boolean operator OR was being used by the search engine. Later on, this problem was sorted out by introducing the Boolean operator AND, available for searching the information on the Web. Due to this the numbers of links in the returned results were decreased. These types of improvements were carried out continuously and brought out into the next generation. These improvements may be inclusion of new technique or updation in the existing techniques. Some main features discovered during earlier generations, include the following:

• The search engine such as AltaVista searched the user given query keywords in title or URL of the Web page (Novak 2003).

• The search engine like Electric Monk, started to use natural language processing. They allowed the user to enter the query in the same manner in which they are desired to find the results. This helped the search engine in finding the desired information.

• The number of hit achieved by a link was also being considered while returning the results to the user. A link with larger number of hits was considered and closer to the user given query. That’s why such types of links got the top position in the returned results of the search engine.

The search engine based on any technique or on any feature provides the results corresponding to the user query in few seconds only. This is because the search engine collects and maintains the information in its repository with the help of a
software program called Web crawler. The general architecture of a search engine is shown in Figure 3.1. The crawler collects the information from different servers of the WWW. The collected information is sorted out, indexed, and stored on the search engine database. Whenever end user searches the information, search engine searches its own database and provides the link of Web documents that may contain the required information. The search engine may present millions of links in response to the user query.

3.3.1 Search Engine Architecture

The search engine architecture consists of the following components: (a) User Interface (b) Searcher (c) Evaluator (d) Web crawler (e) Indexer (f) Repository

(a) User interface: Today, every search engine provides Graphical User Interface (GUI) to search the information. The user initiates this process by submitting his query in the form of words, or phrase on this interface. The search engine searches its local database and returns the ordered results to user in the form of links. All the searchable queries can be classified in three categories: informational queries, navigational queries and transactional queries (Manning et al. 2009). It may also be possible that a query may fall in more than one group.

- **Informational Queries**: This type of queries provides the information about broader area of the topic such as educational institutes etc.. For such type of queries, a single link may not provide the complete information. So, end user must have to visit multiple links.

- **Navigational Queries**: Whenever a user is interested in home page of any Web site then such types of queries are used. This may include queries like Indian Army. Such types of queries show that the user is interested in home page of the Web site.
- **Transactional Queries:** The query that performs only transactional actions lies in this category. For example, online payment of online shopping.

(b) **Searcher:** This component searches the user given query terms in the database of search engine. Searcher walks through millions of indexed Web pages to find the matching document(s).

(c) **Evaluator:** The Evaluator takes the list of URLs returned by the searcher. In the list of these URLs, all the links are not equally important. Therefore, it is good to provide some order to these returned results. This ordering process is carried out by the Evaluator. It is a program that assigns some score to each Web page. The relevancy score is an indication of how well a Web page is matched with the given query. The computation of relevancy score may vary from one search engine to another. For example – a study of (Brin and Lawrence 1998) shows that Google uses the algorithm shown in Figure 3.2 to evaluate the query.
**Algorithm:** Evaluation of Query

**Input:** User given query

**Begin**

Step 1: Parse the given query into words

Step 2: Assign the word ID to each parsed word

Step 3: For every word of document list in the barrels

Step 4: Scan each barrel for the document list till the matching document is found

  **If** (document found) **Then**
  
  Goto step 5

  **Else if** (all the barrels are visited) **Then**
  
  Goto step 6

  **Else**

  Repeat step 4

Step 5: Evaluate the rank of the matched document and store it in a list. Go back to step 4

Step 6: Sort the documents according to the rank score evaluated in the previous step, Higher the rank score then higher the order in the list

Step 7: Provide the ranked list to the user

**End**

**Output:** Ranked list of URLs

---

Figure 3.2: Algorithm for Evaluation of Query

The Step 5 of the algorithm shows that Google uses ranking mechanism for evaluation of the query. The study of (Lawrence et al. 1999) shows that the Page Rank mechanism can provide a good order to the returned results. Many ranking mechanisms (Kleinberg 1999; Diligenti et al. 2002) have been proposed but the ranking algorithm proposed by Sergey Brin and Lawrence Page (Brin and Lawrence 1998) is more popular than others. This algorithm calculates the page rank iteratively and is explained as follows.
\[ PR(A) = (1-d) + d \sum_{i=1}^{n} \frac{PT(Ti)}{C(Ti)} \] 

... (3.1)

Where
- PR is Page Rank
- d is damping factor which can vary from 0 to 1. But generally it is set to 0.85.
- PR(Ti) is page rank of page Ti
- C(Ti) represents the number of outgoing links from page Ti.

This equation can be solved iteratively by taking initial value 1 for each of the Web page. At each step a new page rank value is computed by considering the latest page rank value of that page until the values of page rank converge to a fixed value.

Today, a number of other factors are also considered while calculating the page rank. Some of the factors typically considered here are:

- How many query keywords appear in the Web document
- Whether or not the query words appear in the title.
- How many times query words appear in Web page.
- How many hits a Web page has achieved in the past.

(d) Web Crawler: A search engine maintains its data base of Web pages with the help of a software program called Web Crawler (Cho and Molina 2002). A Web Crawler is also known as a Spider, Bots, Worm or Robot. A crawler is a program that automatically runs at regular interval and returns the pages to the search engine’s local repository (Cho and Molina 2000; Cho and Molina 2000a). Some of the examples of Web Crawlers are: WebCrawler (Pinkerton 1994), UbiCrawler (Boldi et al. 2004), Mercator (Heydon and Najork 1999), WebFountain (Edwards et al. 2001), WebRACE (Zeinalipour and Dikaiakos 2002) etc.
To start the process of crawling an initial set of URLs called Seed URLs are provided to the Web crawler. The Web crawler downloads the Web pages corresponding to the initial URL. After that, each time the crawler extracts the new URLs from the download Web page and adds them to the queue containing the URLs to be crawled. The crawler receives the next URL from this queue in the adopted order. This process is repeated until a stopping condition is reached.

(e) **Indexer**: As soon as the crawler downloads the Web pages from WWW, they must be placed in local repository of the search engine and get indexed. Indexer uncompressed the Web pages available in central repository (Brin and Lawrence 1998) and indexes the Web pages in such a way that searching should be performed speedily and results could be retrieved quickly. Since the resources on the Web are very dynamic and get changed rapidly, hence, indexing of data base needs to be rebuilt again and again as soon as a new page is downloaded.

(f) **Repository**: The Web pages downloaded from the WWW are compressed and then forwarded to the local repository of the search engine for storage (Brin and Lawrence 1998). Every document in the repository is assigned a DocID. This ID is assigned to every newly fetched document. This is only the place where a searcher visits again and again whenever the user enters his query on the search interface.

### 3.3.2 Issues and Challenges with Search Engine

Currently, Web search engines have a number of problems in maintaining or enhancing the quality of their performance. These problems are described as follows:

(i) The search engine tends to examine only the first page of search results. Most of the time only first screen is requested. Thus the inclusion in the first result screen, which usually shows the top ten results, is gradually affected by spam wars or pay-per-click advertising. This significantly changed the landscape regarding a search engine.
(ii) Many issues related with the quality of contents like noisy data, low quality, unreliable and contradictory contents presents a challenge in providing relevant data to end user.

(iii) The search engine try to avoid crawling and indexing duplicate and near duplicate pages but still user experiences such type of problem.

(iv) Cable companies and other Internet Service Providers (ISPs) are lobbying for rule changes that would allow them to block or restrict the access by their users to individual, arbitrarily hand-picked Web sites and other Internet destinations. In such a scenario major search engine is first choice for restriction. ISPs could charge additional charge from search engines for unrestricted access by the users or replace major search engines with their own search services.

3.4 Meta Search Engine
A meta search engine is also called all-in-one search engine. A meta search engine calls multiple search engines (which are publically available) to complete the task of searching. Therefore, it does not require its own database to be created and maintained (Meng et al. 2002). Meta search engine sends the user query to selected group of search engines. The returned results of search engines are combined into a single list (Zhang and Cheung 2003). While preparing this merged list for the returned results, duplicity of results is also eliminated. After all, the resulted list is ranked according to how well they match the user query.

3.4.1 Architecture of Meta Search Engine
The main benefits of using the meta search engine is that a number of different search engines can be accessed by using a single query and a single interface only. So, a large number of matched results can be achieved. The main difficulty of using the meta search is that lot of results returned by the meta search engine are not of user interest. Some of the meta search engines are MetaCrawler (Dwork et al. 2001), Mamma (Mamma 2013), Dogpile (Dogpile 2013) etc. The generalized architecture of a meta search engine is shown in Figure 3.3. Meta search engines enable users to enter search criteria once and access several
search engines simultaneously. This may save the user from having to use multiple search engines separately. Different Meta search engine have different properties and advantages as compared to the other. It may also be possible that the drawback of one search engine is not present in the other or vice versa. So, all Meta search engine are different from one other. Some of the Meta search engine with their description and retrieved result is shown in Table 3.1.

3.4.2 Issues and Challenges with Meta Search Engines
Meta search engines can be very useful in many aspects and they always strive to supply the users need. However, there are some issues and challenges (Patel and Shah 2012; Cutting et al. 1992; Hearst and Pedersen 1996; Kobayashi and Takeda 2000) with meta search engines that should be taken seriously for development of a Meta search engine are as follows:

![Diagram of Meta Search Engine Architecture](image)

Figure 3.3: The Architecture of a Meta Search Engine
Table 3.1: Existing and Available Online List of MSE

<table>
<thead>
<tr>
<th>S. No.</th>
<th>Meta Search Engine</th>
<th>Description</th>
<th>Search Engine from which Links Retrieved</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Excite</td>
<td>The crawler based search engine which has its own index.</td>
<td>Google, Yahoo and Bing.</td>
</tr>
<tr>
<td>2</td>
<td>iBoogie</td>
<td>This Meta search engine has a customizable search type tabs.</td>
<td>All links are from Blekko.</td>
</tr>
<tr>
<td>3</td>
<td>Fuzzyfind</td>
<td>It has trends in pictures and news feed sections.</td>
<td>All links are from Google.</td>
</tr>
<tr>
<td>4</td>
<td>Apollo7</td>
<td>The Meta search engine that search result from customizable list of search engine.</td>
<td>AlltheWeb, Bing, Curryguide, Google.</td>
</tr>
<tr>
<td>5</td>
<td>Search</td>
<td>The Meta search engine operated by CNET (tech media Website that publishes review, news etc).</td>
<td>Google, Blekko and Bing</td>
</tr>
<tr>
<td>6</td>
<td>DuckDuckGo</td>
<td>In this the privacy to user query is the main concern for user</td>
<td>Various search engines</td>
</tr>
<tr>
<td>7</td>
<td>Scandoo</td>
<td>Meta search engine based on only 4 search engine.</td>
<td>Google, Yahoo, Bing and Ask</td>
</tr>
<tr>
<td>8</td>
<td>Carrot2</td>
<td>It has various links and diagram to help in search result</td>
<td>Mostly from Google and some from Ask, Bing and Yahoo</td>
</tr>
<tr>
<td>9</td>
<td>Dogpile</td>
<td>Meta search engine that displays results from each search engine individually</td>
<td>Google, Yandex and Yahoo</td>
</tr>
<tr>
<td>10</td>
<td>Gnome</td>
<td>It is the Meta search engine, portal and directory</td>
<td>From 10 popular search engine</td>
</tr>
<tr>
<td>11</td>
<td>Web Crawler</td>
<td>It is the crawler based Meta search engine</td>
<td>Yahoo! search, Google</td>
</tr>
<tr>
<td>12</td>
<td>Meta Crawler</td>
<td>It is the Meta search engine</td>
<td>Google, Yandex and Yahoo</td>
</tr>
</tbody>
</table>
Enhance the search coverage of the Web: A study of (Lawrence and Giles 1999) indicates that the individual search engine cannot cover more than 16% of the entire Web. All this happens due to the exponential growth of the Web documents rather than the indexing capability of any single search engine. This problem of coverage can be solved by using multiple search engines through a meta search engine. As each search engine can have different coverage of the Web; so, a much higher percentage of Web coverage can be achieved by using a meta search engine.

(ii) Facilitate the use of multiple search engines on a single platform: Any information required by a user is stored in the databases of multiple search engines. As an example, consider the case when a user wants to find the information about the Kurukshetra University, Kurukshetra. It is possible that the information/articles are available from multiple search engines. But, a user can see this information only after visiting these returned links. So, for single information a user has to request multiple search engines. This makes it very difficult task for searching the information because:-first, user has to identify the sites containing the information related to the Indian prime minister; second, the user has to send the query to each of these search engines. As different search engine accept the user query in different format. So, the user has to formulate his query accordingly; third, the quality ranking of different search engine are not same. So, a link available on first page of a search engine may not be present on the first page of other search engine or vice versa. Therefore, it becomes difficult to determine which article will be most useful ones. That’s why; a Meta search engine should have a mechanism that can rank the returned documents properly.
(iii) **Effective Presentation of Results:** MSE combines the results of multiple SEs together on a single platform. But currently available MSEs are not able to present the results in an effective manner.

(iv) **Pre – Determined Number of Clusters:** Number of clusters required for the results returned by the SE can be decided before downloading the Web pages. But post retrieval clustering of documents can provide better results.

(v) **Ir-relevant Cluster Names:** MSE like Clusty provide the results to the user in the form of named clusters i.e. it organizes the links of SEs in different folders. The name of these folders is set automatically to the word which has the highest frequency in the Web page. But the name assigned to these folders may not be relevant to the keywords searched by the user, as it just picks up the highest frequency word irrespective of whether they match or do not match with the query keywords.

(vi) **Ir-Relevant Web Pages:** MSEs like Excite put a link on the top of the ranked list if the link is retrieved by multiple SEs. But a link retrieved by multiple SEs cannot be straightaway declared as the most important link. According to (Yon and Dong 2009), a link retrieved only by a single SE has been found to be more relevant than those found by multiple SEs.

(vii) **Poor Quality of Web Pages:** According to (Kobayashi and Takeda 2000), 85% Internet users uses Internet for topic specific queries but most of them are found unsatisfied with the returned results of SE. This is because; they do not undertake the concept of relevancy.

(viii) Ranking algorithms working on the basis of ‘Positional Ranking’ and ‘Count Functions’ do not deal with the relevancy.
3.5 Crawling the Web

Crawling is the way through which the Web crawler collects the Web pages from the World Wide Web. The final outcome of the crawling process is the collection of Web pages in the repository of search engine. The Web is continuously expanding with each passing day, so, the data available with the search engine repository is also expanding. That’s why a crawler has to crawl lot of Web pages daily.

A crawler identifies the location of a document on the Web by its URL. The crawler can download the Web page from this URL. Figure 3.4 shows the complete process used by the crawler for downloading the Web page. The crawler maintains a queue called Frontier containing the list of URLs to be visited. This list is initialized with seed URLs which may be provided by a user or by any other program. The crawler will download the Web pages by taking the URL from this queue. The crawler also extracts the URLs (if present) from the downloaded Web page and place them in Frontier. After that the crawler applies some ranking mechanism to set the priority of these links. Crawler then chooses the next URL with highest priority from the Frontier queue and crawl this URL on WWW. This crawling process may be stopped when a certain number of pages have been crawled. There may be a situation in which the crawler is ready to crawl but the frontier is empty.

In this situation, a signal called dead-end is generated for the crawler. As soon as the crawler receives the dead end signals it automatically understands that there are no further URLs to be crawled.

The working of a crawler may be explained with the help of algorithm shown in Figure 3.5. The crawler remains on one station and downloads the Web page from WWW. The crawler does not move from one station to another as mobile agent or viruses do (Peshave and Kamyar 2005).
Whenever the user directly provides URL to address bar of the browser or clicks on a particular link then Crawler sends a HTTP request to some other computer on the Internet. It must be noted that other computer should also be HTTP enabled. The basic flow control to understand the functioning of a Web crawler is shown in Figure 3.6.
Algorithm: Basic Crawling Process

Input: Seed URL

Begin

Step 1: Provide the Seed URL(s) to Frontier Queue Q

Step 2: While (Q is not empty)

Retrieve the top URL from Q and download the corresponding WP

Step 3: Extract the URL(s) from the downloaded WP and store the downloaded WP into central repository of SE

Step 4: Store the extracted links in Frontier //according to their priority

Step 5: Repeat this process till the frontier does not becomes empty

End

Output: Downloaded Web page in repository and extracted URL in frontier

Figure 3.5: Algorithm for the Crawling Process.

3.5.1 Basic Component of Web Crawler

The main components (Sharma et al. 2003) of a Web crawler are: Frontier Queue, Web Page Repository with History, Fetcber and Parser. These components are described in sufficient detail below:

(a) Frontier Queue: It contains the list of unvisited URL(s). The URL in this list may be complete path or can be a partial path of the Web document. The URL stored in this queue may or may not be present on the Web. It was assumed that there are near about 60,000 URLs with every 10,000 crawl of Web pages. This estimation was based on the assumption of seven URLs per Web page. The URL extracted from frontier comes in FIFO order. So, one can say that next URL to be crawled comes from the head of the Frontier queue and new URL is added at the rear of the Frontier Queue.
Figure 3.6: Flow Control for a General Web Crawler
(b) **Web Page Repository with History:** Web pages are stored in the central repository of the search engine with the history of Web page. This history may include time when the Web page was last modified, number of times a link got hit, etc. Such types of data are maintained for the analysis and evaluation purpose. For example, from such types of data one can easily identify the age of the Web page.

(c) **Fetcher:** Fetcher is used to fetch the Web page from the WWW. The Web page to be fetched is identified by its URL only. An HTTP request is provided to the Web browser with the URL of the Web page. The requested Web page can be downloaded only if it is available for public access. So, there is a protocol on the Web called Robot Exclusion Protocol which indicates whether a crawler can access a particular file or not. The file containing such types of permissions is called Robot Exclusion Policy. This file is maintained with the name of robot.text under root directory of the Web server.

(d) **Parser/Extractor:** Every Web page is scanned as soon as it is fetched from the WWW. The Web page is scanned to check whether the fetched page contains any link to other page or not. This process is accomplished by making the HTML tag tree. The tag tree does not contain any stopping word such as “\it”, “\for”, “\of” etc. This process of removing the stopping words is called ‘stop listing’.

### 3.5.2 Features of a Web Crawler

A Web Crawler should have the following features (Cho and Molina 2002):

(a) **Novelty of Web pages in Central Repository:** As the WWW is very dynamic in nature, so, a crawler should crawl or visit the Web page repeatedly to maintain the central repository up-to-date. This helps the searchers or end users in finding the latest information from the WWW.
(b) **Quality of Web Document:** As the size of WWW is very large, so, it is not possible to crawl and index each and every Web page available on the Web. That’s why; it becomes necessary to download the relevant and important Web page first. Therefore, some effective and efficient techniques should be adopted to download good quality Web pages.

(c) **Support for Distributed Environment:** The documents available on the Web are in unstructured form and the working environment may vary from one machine to another. So, a Web crawler should be able to work on different and distributed environment.

(d) **Support for Performance and Effectiveness:** A Web crawler should be able to utilize all the available resources up to their maximum level. These resources may include utilization of CPU, storage of Web pages and bandwidth of network.

(e) **Extensible:** A crawler should be designed in a modular fashion so that it would be able to work on different data formats and different protocols, etc.

(f) **Scalable:** The provision of expansion should also be present in a Web crawler. The expansion might be in the form of some extra machines or expansion in network bandwidth.

(g) **Respect for Server Policies:** Each and every server available on the Web has its own policies for accessibility. The standard of these policies may vary from one server to another. The policies of some of the servers may allow for accessing some or all the available Web pages while others may keep it as hidden. Therefore, a crawler should follow these policies of the servers very gently.
3.6 Web Crawler: Issues and Challenges

Due to large size of the World Wide Web, it is not possible for a single Web crawler to download more and more Web pages with in the given time limit. Also, a single crawler is not able to see and check the content and structure of the Web page. Hence different types of Web crawlers are in use today. The expertization of these crawlers may vary from one crawler to another crawler. However, large amount of data is available on the Web to be retrieved but there is one common aspect of crawling, which is continuously being improved i.e. efficiency. The classification approach classifies the Web crawler in four categories i.e. – Basic Crawler, Focused Crawler, Parallel Crawler and Distributed Crawler. Each of these crawlers is described in the following sections:

3.6.1 Basic Crawler: The basic crawler retrieves all the data without any discrimination. It is very simple to implement and becomes a basis for other crawlers going to be developed. In the beginning, the development of crawlers was based on tradition graph algorithms such as breadth first search or depth first search to index the Web. A set of seed URLs are used and algorithm recursively follows all the hyperlinks down to other documents. The traditional graph algorithm differs in the way in which they retrieve the Web pages but the end result of both algorithms obtains the same set of Web pages. With the basic crawling a large number of Web pages are stored in central repository with some ranking mechanism. When a user searches anything through keywords then the results were provided to the user from this central repository. In the early days of search engine development, a basic crawler was sufficient.

3.6.1.1 Issues and Challenges with Basic Crawler: However, as the time passed the size of Web also increased significantly; it has produced the following issues and challenges towards the efficiency of the basic crawler:
(a) **Scalability:** A search engine can cover only 30% of the entire Web to download and store the Web pages in the central repository. But to refresh these Web pages takes weeks to months (Lawrence and Giles 1998). Therefore, the basic crawler is not able to synchronize the central repository with the rapid and dynamic growth of the Web.

(b) **Information Overkill:** A search engine with a basic crawler provides a long list of results in response to the given query. Furthermore, these results contain a lot of irrelevant Web pages in the result. The Web search engine tries to cover complete Web and download the query relevant Web pages (Ester et al. 2001). In fact, the number of results returned by the search engine is not a matter of interest because large number of returned results leads to the problem of information overkill.

(c) **Database Size:** One for all purposes crawler is not possible to implement because a single crawler cannot cover the whole Web but can index billions of Web pages. Most of these Web pages would never be used either by the end user or crawler but the burden of maintaining the huge database is always there.

(d) **Network Traffic:** The basic crawler downloads the Web page corresponding to each available link. But it is not always possible or required to download the Web page corresponding to each link. Hence, basic crawlers have remarkable pressure on the network traffic.

To overcome the above limitations of the basic crawler, a new model called focused crawler was designed. Focused crawler downloads only selected portion of the Web instead of downloading the whole desired and undesired data.

3.6.2 **Focused Crawler:** The Focused Web crawler retrieves only a set of selected document from the WWW and avoids retrieving less significant Web pages. This results, in minimization of the wastage of resources and it maximizes
the chances of downloading important pages only. In this way Focused crawler concentrates on quality of information. The concept of the focused crawler was given by S. Chakrabarti (Chakrabarti et al. 1999) in 1990’s but it got more attention only in the recent years (Diligenti et al. 2000; Qin et al. 2004; Papapetrou and George 2004; Menczer et al. 2004; Shokouhi et al. 2005; Su et al. 2005; Zhuang et al. 2005; Pant and Srinivasan 2005; Pandey and Olston 2005; Gao et al. 2006; Pant and Srinivasan 2006; Mohsen et al. 2006; Álvarez et al. 2007; Zheng et al. 2008; Huang et al. 2009:). A Focused crawler analyzes its boundary to crawl the links that are likely to be most relevant for its crawling. The decision to crawl a Web page is based on the information available about the page before retrieval.

This information is generally obtained from the anchor text of the links. It explores the Web with best first search technique according to a specific topic. Thus, a Focused crawler predicts it in prior that whether the target URL is pointing to a relevant and high quality document or not. The Focused crawler does not collect all reachable Web pages as the basic crawler does. Hence, it downloads only topic relevant documents in its path (see Figure 3.7) instead of downloading all the links as in case of a basic crawler (discussed in section 3.6.1).

3.6.2.1 Focused Crawling Techniques: The Focused crawling techniques for retrieving the relevant and important Web pages have been classified in two categories i.e. lexical based approach and link based approach. Each of these techniques is described as follows:

(a) **Lexical Based Approach:** This approach analyses the information directly from the contents available in the HTML Web page (Mishra 2012). For example, the relevance of the Web page can easily be determined by looking the text available in the title. Mostly, a higher weight is assigned to text available in title
or heading of the HTML page. Knowledge of domain also plays an important role into the process of improving the results. For example, the words available in a Web page can be analyzed based on the domain specific terminology. In such types of systems the terms available in a Web page are compared with the terms available in list of the domain specific terminology.

![Diagram](image)

**Figure 3.7: Graph for Focused Crawling**

**(b) Link Based Approach:** Link Based Approach (discussed in section 3.7.1) uses link structure of a Web page to check the importance of a Web page. It becomes challenging when the page pointed to by many Web pages does not lead to another popular page.

The main advantages of using the Focused crawler are as follows:

(i) The search engine that uses the Focused crawler can have more important and relevant Web pages in its repository as compared to basic crawler, even though they start from the same seed URL.
(ii) It can discover only valuable links from a large list of links. That may result in high collection of Web pages on a specific topic.

(iii) The focused crawling is able to avoid the overlapping of Web pages.

(iv) The focused crawler is also able to identify the regions of the Web that grow or change radically as against those that are relatively stable.

3.6.2.2 Issues and Challenges with Focused Crawler

The following four issues were found with the focused crawlers:

(i) **Missing Relevant Pages:** One issue with focused crawlers is that they miss relevant pages by only crawling pages that are expected to give immediate benefit.

(ii) **Maintaining Freshness of Pages:** Many HTML pages consist of information that gets updated on daily, weekly or monthly basis. The crawler has to download these pages and update them into the database to provide up to date information to the users. The crawling process becomes slow and puts pressure on the Internet traffic if such pages are large in number. Thus, it is a major issue to develop a strategy that manages these pages.

(iii) **Network Bandwidth and Impact on Web Servers:** The focused crawling techniques download many irrelevant Web pages that lead to consumption of network bandwidth. They adopt polling method or deploy multiple crawlers for maintaining the freshness of database. Both methods consume lot of bandwidth. Also, there is crippling impact on the performance of Web servers if the crawlers are visiting them for all information. Thus another issue is to develop some method to retrieve only highly related pages and alternate technique to poll the Web server so that the underlying resources are not overloaded.
(iv) **Absence of Particular Context:** The focused crawler uses the best strategy to down load the most relevant Web pages based on some defined criteria. The crawler focuses on a particular topic but in the absence of particular context, it may download large number of irrelevant pages. Thus the challenge is to develop focused crawling technique that focus on particular context also.

3.6.3 **Parallel Crawler:** The growing size of World Wide Web makes it impossible to crawl all available Web pages by a single process. So, a search engine can run multiple crawlers in parallel to maximize the download rate and to retrieve the whole or significant portion of the Web. This type of crawler is referred as a parallel crawler. For parallel crawling, an augmentation to hypertext documents in the form of Table of Links (TOL) for parallel crawling was developed by (Sharma et al. 2003). In this proposal the crawler needs to know about the owner of the document, modification date of the document, date of expiry for the document, the links to other documents on the same site, the links to other documents on the other sites across WWW. According to this proposal - if the URL contained within a document becomes available to the crawler before the instance when the crawler starts downloading the documents itself, then downloading of its linked documents can be carried out in parallel by other instances of the crawler. But in the current scenario, the links become available to the crawler only after a document has been downloaded by the downloader. TOL consists of the links contained in a document. The TOL is stored within the document in such a way that it can be retrieved and parsed separately by the crawler. The TOL is encapsulated within a container object so that it becomes able to provide the information on demand to the crawler. The storage may be in the form of a file with the same name as the document but with different extension (say as .TOL). TOL is also identified three challenges for changes in a document as follows:
(i) If the document has been moved to some other location i.e. its URL is redirected.

(ii) What happens if, the document cannot be accessed because of broken links?

(iii) The contents of some documents get changed periodically or the contents of a document get changed on their expiry date.

According to TOL, first and second problems can be handled at server and crawler level whereas the third problem cited above can be tackled at the document level. Some HTTP servers transmit the modification date with the document itself, which reduces the load on the network. Authors of the proposal include a new tag called Periodicity Tag in the HTML documents which indicates the number of days after which the page will be changed or modified. This would be stored in the crawler’s database after the document has been downloaded. The crawler (design in progress) would analyze such URL’s on daily basis to identify the potential sites liable to change on that day so that the documents can be refreshed on a priority basis. This would ensure that the crawler always has fresh pages.

3.6.3.1 Features of Parallel Web Crawler: Although the parallel crawlers are difficult to operate, yet they have some advantages as compared to a single-process crawler. These advantages are as follows:

(i) **Scalability:** Due to the huge size of the Web, it may be advantageous to run a parallel crawler to achieve the required download rate in certain cases.

(ii) **Network-load dispersion:** Multiple crawling processes of a parallel crawler may run at geographically isolated locations. Each process downloading “geographically-adjacent” pages only. For example, a process in India may download all Asian pages only, while another process in Germany may
download all European pages only. In this way, the load of the network can be scattered to multiple regions. This policy of load distribution becomes essential when a single network cannot handle the heavy load from a large-scale crawl.

(iii) Network-load reduction: A parallel crawler is also able to reduce the load on the network. For example, assume that a crawler in any country of Asia retrieves a page from any country of Europe. In this case, the page first has to go through the network in Europe, then the Europe-to-Asian network and finally to the crawler. Instead, if a crawling process in Europe collects all European pages, and if another process in an Asian country crawls all Asian pages, the overall network load will be reduced, because pages only go through “local” networks. It must be noted that at the last stage, the downloaded pages may need to be transferred to central repository. But this transportation of Web pages may not contain all of the downloaded Web pages because sometimes only a significant portion of a Web page is sent to the central repository by using some of the following methods:

- **Compression:** As soon as the Web page is downloaded, it can be compressed easily and then sent to central repository for storage.

- **Difference:** Instead of sending the entire Web page to the central repository, one could take the difference between the previous and the current image of the downloaded Web page and send only this difference to the central repository. Since many pages are static and do not change very often, this scheme may significantly reduce network traffic.

- **Summarization:** In certain cases, there could be a necessity of central index only, not the original pages themselves. In this case, only necessary information is extracted from the Web page and only this information is transferred to the central repository.

The general architecture of parallel crawlers is shown in Figure 3.8.
Figure 3.8: Architecture of Parallel Crawler
The individual crawlers working in this architecture may be located in same local area network known as intra-site parallel crawler. It may also be possible that it may be located in a geographically distant location connected through Internet, as distributed parallel crawlers.

3.6.3.2 Issues and Challenges with Parallel Web Crawler: The following issues make the study of a parallel crawler difficult, yet interesting:

(i) **Overlapping of documents:** While working in parallel, it may be possible that multiple crawlers download the same pages many times because they may not be aware about which pages have already been downloaded.

(ii) In this way, there is wastage of resources like network bandwidth etc., it therefore, reduces the efficiency of a parallel crawler. Therefore, methods are needed to avoid such types of overlapping.

(iii) **Quality of Downloaded Web Pages:** As discussed in section 3.2.2, the main objective of any crawler is to download Web pages first. While working in parallel each crawler may not be aware about the whole collection of the Web pages downloaded collectively. Their crawling decisions may be based on their local collection of Web pages and this may lead towards poor crawling decisions. Therefore, there should be a mechanism to make sure that the quality of downloaded pages should be as good for a parallel crawler as for a single-process crawler.

(iv) **Network Bandwidth:** To minimize the overlapping of documents and to enhance the quality of downloaded Web pages, individual crawlers need to communicate among themselves to coordinate the whole crawling process. But the drawback with this communication approach is that it will consume bandwidth as well as crawler’s time. Hence, there should
be a proper strategy to minimize the communication while maintaining the quality of crawling.

3.6.4 Distributed Crawler

The Distributed Crawler is an extension to a parallel crawler. The distributed crawler is different from the parallel crawler in the sense that multiple processes are isolated geographically instead of being on same machine or group. Ubicrawler (Boldi et al. 2002), Mercator (Najork and Heydon 2001) and Web Fountain ((Edwards et al. 2001) are the examples of distributed Web crawlers.

3.6.4.1 Architecture of Distributed Crawler

The architecture of Distributed crawler is shown in Figure 3.9 in which the crawler design is separated into two main components, referred to as crawling application and crawling system. The crawling application decides what page to request next given the current state, previously crawled pages and issues a stream of URL requests to the crawling application for analysis and storage.

![Figure 3.9: Architecture of Distributed Crawler](image_url)

The crawling system is in charge of tasks such as robot exclusion, speed control, and DNS resolution, while the application implements crawling strategies. The crawling system consists of several components that can be replicated for higher
performance. Both crawling system and application can also be replicated independently and several different applications could issue request to the same crawling system. The use of distributed crawler has the following benefits:

(i) Crawlers can share different networks; therefore, crawling throughput would not decrease with increase in crawling nodes.

(ii) Crawler on different geographical locations can perform local crawling on nearby sites which decreases the traffic load on the network. So now, a separate HTTP request would not require for fetching the data from remote system using mobile agents.

(iii) A failure (like power failure or natural disaster) would not be able to affect the whole crawling process.

3.6.4.2 Issues and Challenges with Distributed Crawler

The issues and challenges available with the Distributed crawler are as follows:

(i) **Assignment of URL’s among different agents:** The major challenge in distributed crawler is how the URLs are assigned efficiently and dynamically to download among the crawling agents. The assignment must take into consideration the various constraints like the rate of requests to Web server should be minimize, the crawling agents should be located appropriately on the network, and the URLs should be exchanged effectively.

(ii) **Priority in Crawling:** There is no proper way to efficiently prioritize the crawling frontier considering the dynamic nature of the Web.

(iii) **Effective way of Partitioning the Collection:** The effective way of partitioning the collection in such a way that the query answering phase worked by querying not all partitioning, but only the smallest possible set of the partitions. The chosen subset should be able to provide a high number of relevant documents.
(iv) **Load Balancing:** Determining the effective way of balancing the load among the different index servers is a big problem. There must be a good strategy to distribute the data in order to balance the load as much as possible.

(ii) **Network Bandwidth Consumption:** Network bandwidth is a scarce resource and is a big challenge. Therefore, when queries are resolved in a distributed fashion, the servers that should be connected must be determined efficiently. Also the number of necessary of servers should be limited.

### 3.7 Crawling Policies

The main objective of any Web crawler is to download the most important Web page from WWW. As the size of WWW is very large, therefore, it is not possible for one crawler to download all the available Web pages in a restricted time. Several policies have been proposed in (Cho et al. 1998) to check the importance of Web pages. This helps the Crawler in determining the importance of a Web page. Based on this importance value, a crawler can decide whether to download a given page or not.

#### 3.7.1 Similarity Based Crawling Policy

The similarity between the Web document (d) and the query (q) can be viewed as an m-dimensional vector \(<w_1, w_2, ..., w_m>\). The \(w_i\) represents the \(i^{th}\) word in the vocabulary. If it does not appear in the vocabulary then a weight zero is assigned to it, otherwise some weight is assigned to it. The significance of \(w_i\) is computed by multiplying the frequency count of \(i^{th}\) word with Inverse Document Frequency (IDF) count. IDF is calculated by dividing 1 by the frequency of word in the entire collection. Here, the entire collection signifies the complete Web. But, it is not possible to crawl the entire Web; that’s why only the crawled Web pages are taken into consideration. It also considers the rarely found words like “queue” and generally found words like “the” while providing the weight to a
Web page. After that, textual similarity can be defined as inner product of Web document ‘d’ and query ‘q’.

3.7.2 HITS Based Crawling Policy

A connectivity based algorithm for hyperlinked environment is proposed in (Kleinberg 1999) to rate a Web page. The set of search engine results is given to the HITS algorithm as an input. HITS extract the sub graph from the Web containing the result set. Then iterative computation is performed to estimate the value of each document as a source of relevant link and as a source of useful content. HITS (Hyper–Link-Induced-Topic Search) algorithm identifies two types of links – Authority pages and Hub pages as shown in Figure 3.10.

- **Authority Pages:** The high quality pages related to a particular topic or search query.
- **Hub Pages:** The Web pages that provide a pointer to authority pages.

Based on this, every Web page on the Web is associated with an Authority score and a Hub score, which are used to identify the Web page context. HITS maintain an augmented set for each page p, a hub score h(p) and an authority score a(p).

---

**Figure 3.10: Hubs and Authority in HITS**

103
Each iteration of HITS consists of two steps: step 1 replaces each a(p) by the sum of h(p) values of page pointing to p; the step 2 replaces each h(p) by the sum of the a(p) values of the pages pointed to by p. The Authority score of a page p depends on the Hub score of pages pointing to p (H₁ to Hₙ). Similarly, the Hub score of page p depends on the authority score of the page p pointing to (A₁ to Aₙ). A good authority increases the hub weight of the pages that point to it. So, a good Hub page points to many good Authority pages. A good Authority page is pointed to by many good Hub pages.

3.7.3 DOM Based Crawling Policy

Integration of hyperlink and Document Object Model (DOM) of Web pages for topic distillation is proposed in (Chakrabarti et al. 2002). When DOM parsers successfully parse an XML document then it creates a tree structure in memory that contains the data of document. In this method, a Web page is viewed as tree with <HTML> as it’s root and different tags and text forming the tree nodes as shown in Figure 3.11. DOM tree structure available in memory can be used easily to extract the tag and data quickly. It is assumed that every HTML page has a DOM tree. The leaf nodes of this tree are HREFs to the nodes of other DOM tree. Thus, this method is able to identify and extract hub regions relevant to a topic and may also guide the hub and authority for reinforcement of a selected sub graph of the Web.

3.7.4 Page Rank Algorithm

Page rank provides a numeric value to each and every page on the Web. This value provided by the page rank indicates the importance of the Web page. The importance of Web page p is recursively defined as weighted sum of back links to it. This algorithm provides larger value to more important Web pages and smaller value, to less important Web pages. The details about Page rank are already discussed in section (3.3.1).
3.7.5 Location Based Policy

In Location based policy the importance of a Web page depends on location of words rather than its contents. For example, the URL of a Web page ending with .com extension is considered as more useful than the URL ending with any other extension.
Similarly, small URLs are considered more important than larger ones. All these parameters are called as local parameters because they can be evaluated by simply looking at URLs only.

3.7.6 Link Based Policy

The Link based policy (Chakrabarti et al. 1998; Chakrabarti et al. 2001; Menczer 2004) has a basic assumption about the Web page i.e. if the author of a Web page “A” includes a link to other Web page “B”, it means that he/she believes that “B” is relevant or similar to “A”. The term in-link is used to indicate the hyperlinks pointing to a given page. Generally, if a Web page is pointed to by large number of other Web pages, then it is considered to be important. It is similar to the citation analysis in which a frequently cited article is considered more important than a never cited one.

3.8 Crawling Algorithms

The major objective of any search engine is that its crawler should download and store more and more but important Web pages in its repository. So, a Web crawler should be designed in such a way that it can easily differentiate between different Web pages. The algorithms such as Depth First Search, Breadth First Search, Best First Search, Fish search and shark search etc. are available to fulfill this goal. In this section, these crawling algorithms have been discussed in brief.

3.8.1 Depth First Search (DFS) Algorithm

In DFS, the crawling start from the initially Web page and it explores all other Web pages that are reachable from this Web page. Then it chooses one link out of the explored Web pages and follows it. It repeat this process for every available Web page and keeps retrieving the Web pages, going deeper and deeper till no further Web page is available to be retrieved. It then back tracks to
the next unvisited node and then continues in similar manner (Shen 2010). The whole process of DFS is explained with the help of Recursive Algorithm given in Figure 3.12. DFS is mainly used for search problems, but when a large number of URLs is available then this algorithm falls in an infinite loop (Coppin 2004).

**Algorithm:** Depth First Search (Web Page)

**Input:** Web Page.

**Notations used:** WP for Web Page, li for URL, In for next URL.

**Begin**

Step 1: Take initial seed URL
Step 2: Download the WP corresponding to this URL and mark it as visited.
Step 3: For (all li in WP)

- **If** (li is unexplored) **Then**
  - Give it to the rear of frontier
- Else **If** (In of frontier is unexplored)
  - Then mark it as Visited and Recursive call to DFS (In)
- Else
  - Mark li as Visited

Step 4: Mark WP as explored

**End**

**Output:** Labeled Web Pages as Visited unvisited and explored.

Figure 3.12: Algorithm for Depth First Search
3.8.2 Breadth First Search (BFS) Algorithm

The BFS algorithm (Cho et al. 1998; Menczer et al. 2002; Srinivasan 2004) performs the crawling across the Web page level by level. The crawling in BFS starts from the initial Web page and then it explores all the available URLs on that page only. As soon as all the links available on first level are explored then it starts exploring all other URLs available on the Web page of next level (as shown in Figure 3.13).

![Diagram of BFS algorithm]

Figure 3.13: Breadth First Search Crawling
The number of Web pages to be explored can vary from one level to other. This process is repeated till no more URL is left to be searched. The whole process of BFS can be explained with the help of algorithm as shown in Figure 3.14.

**Algorithm:** BFS (Seed URL)

**Input:** Seed URL

**Begin**

Step 1: Take initial URL mark it as Visited and store it in Visited Queue
Step 2: Download this URL
Step 3: Explore and extract all available URL from this downloaded WP
Step 4: Add extracted URL to Frontier and store downloaded WP in central repository
Step 5: Extract next available URL from Frontier
Step 6: **If** (extracted URL marked as Visited or available in Visited Queue)

Then

**Goto** Step 5

Else

**Goto** Step 2

**End**

**Output:** Visited URLs, Unvisited URLs and Web pages

Figure 3.14: Algorithm for Breadth First Search

**3.8.3 Best First Search Algorithm**

Best First Search algorithm is a general problem solving approach used in Focused Crawlers (Chakrabarti et al. 1999; Qin et al. 2004) and game software’s. It maintains two lists called Open and Close. Open list contains the list of unvisited URLs and Close list contains the list of visited URLs (see Figure 3.15). The algorithm applies some heuristic technique to choose the best URL
from all available URLs (Quin and Chen 2005). So, this algorithm downloads important and relevant Web pages only (see Figure 3.16). Crawler implementation via this algorithm treats each Web page as a node of graph.

A crawler that uses best-first search, generally, uses an evaluation function that assigns priority to each link is based on how closely the contents of their parent page resemble the search query (Menczer et al. 2001). In games, best-first search may be used for path-finding, For example, it could be used by an enemy agent to find the location of the player in the game world.

Figure 3.15: Best First Search Crawling
**Algorithm:** Best FS(Seed URL)

**Input:** Seed URL

**Begin**

Step 1: Place the seed URL in Open list

Step 2: Fetch URL from Open list, Download the corresponding WP from WWW and store this URL in Close list

Step 3: **If** (Open is empty) **Then**

stop crawling

**Else**

Extract the best URL from the Open list

Step 4: Explore this WP for new URLs.

Step 5: **If** (explored URL is goal) **Then**

Return success with WP and Goto Step 7

**Else**

Apply heuristic function and store it in Open list

Step 6: **Goto** step 2

**End**

**Output:** Open list for unvisited URLs and Close list for visited URLs

---

**3.8.4 Fish Search System**

Fish Search System (FSS) proposed in (Bra et al. 1994), can fetch the Web pages on the principle of school of fish metaphor in the sea. According to this principle, generally a school of fishes moves in the direction of food. While swimming, this school can also split in different schools/groups. So, the number strength of fishes in a group depends on the availability of food. An individual group of fishes that go into direction in which no food is to be found will lead towards their dead end or when the water is polluted they may die.
Here in case of Fish Search System of Web crawling algorithm (shown in Figure 3.17), a fish is search agent, food of fishes is relevant information, and polluted water means poor bandwidth. This algorithm takes a seed URL as an input and then dynamically builds the priority list of next URLs to be explored. After this step, each time the link from this list is popped out and processed. As soon as the text of each document becomes available, the relevance of each Web page is computed. Based on this relevance, a heuristic decides whether to pursue in this direction or not?

Whenever the text of document is fetched then it is searched for the availability of links. The Web pages pointed to by these links are called child and assigned a depth value. If the parent link is relevant, the depth of the children is set to some predefined value. Otherwise, the depth of child is set to be one less than the depth of the parent. As soon as the assigned value reaches to zero, the direction of search is dropped and none of its children is inserted into the list.

Fish-search algorithm is simple but poses following limitations:

(i) It assigns relevance score in discrete values i.e. value 1 for relevant and 0 or 0.5 for irrelevant using regular expression match.

(ii) It is not able to differentiate the priority of Web page available in the list.

(iii) If large number of pages have the same priority and if time constraint is there with the crawler then the documents which are at the tail of the list remain un-crawled and thus there may be loss of valuable information. To reduce these limitations Hersovici, M. et al have proposed Shark Algorithm.
Algorithm: FSS(seed URL)

Input: Initial URL and Depth

Begin

Step 1: Place and take the seed URL from the list

Step 2: While (list is not empty or time limit exists)

Step 3: Download WP corresponding to the URL coming from list.

Step 4: Scan and Parse the downloaded WP for relevant information. Also, extract the external URLs (if available) pointed to by this Web page.

Step 5: If (depth > 0) Then

If (downloaded WP is irrelevant) Then

For (each child of currently downloaded WP)

Set score = 0.5

For (each rest of the children of current WP)

Set score = 0

Else

For (each child of current WP)

Set score = 1

For (each rest of the children of current WP)

Set score = 0

Step 6: For (each child of current WP)

If (child already exists in the priority list) Then

a. Compute the maximum between the existing score in the list to the newly computed score

b. Replace the existing score in the list by that maximum

c. Arrange the new child to its correct location in the sorted list if necessary

Fish Search System – algorithm cont…
Else

Insert child at the rear end of the sorted list according to its score.

Step 7: For (each child of current WP)

If (current WP is relevant) Then

Set depth(child) = D

Else

Set depth(child) = depth(current WP) – 1

If (child already exists in the priority list) Then

(a) Compute the maximum between the existing depth in the list to the newly computed depth

(b) Replace the existing depth in the list by that maximum

Step 8: Goto step 2

End

Output: Sorted and Prioritized URL in list

Figure 3.17: Algorithm for Fish Search System

3.8.5 Shark Algorithm

The Shark algorithm explained in (Hersovici et al. 1998) provides a fuzzy score between 0 and 1 to all the relevant and irrelevant Web pages. Fuzzy relevance score assigned to children helps in selection of a Web page that has a better score. This assigned inherited score may propagate to the descendants of the Web page and thus increases the importance of the grandchildren of a relevant node over an irrelevant node. The modifications that were required in FSS are done in shark algorithm as shown in Figure 3.18.

These modifications are to be done in Step 5 of Fish Search System algorithm (discussed in section 3.8.4).
Compute the inherited score of child, as follows:

\[
\text{If (relevance score of current Web page > 0) Then}
\]

\[
\text{inherited score(child) = \delta \times \text{sim}(q, \text{current Web page})}
\]

//where \( \delta \) is a predefined constant.

\[
\text{Else}
\]

\[
\text{Inherited score (child) = \delta \times \text{inherited score (current Web page)}
\]

The relevance score for textual contents of the anchor pointing to child is calculated as follows:

- Let: anchor_text be the textual contents of the anchor pointing to child, and anchor_text_context be the textual context of the anchor.
- Compute the relevance score of the anchor text as anchor_score = \text{sim}(q, anchor_text)

Computation of relevance score of the anchor textual context as follows:

\[
\text{If (anchor_score > 0) Then}
\]

\[
\text{anchor_context_score = 1}
\]

\[
\text{Else}
\]

\[
\text{anchor_context_score = \text{sim}(q, anchor_text_context)}
\]

Neighboring Web page score denoted by neighborhood_score and can be calculated as follows:

\[
\text{neighborhood_score} = \beta \times \text{anchor_score} + (1-\beta) \times \text{anchor_context_score}
\]

//where \( \beta \) is a predefined constant

Child score can be calculated as:

\[
\text{score(child) = \gamma \times \text{score(childe)} + (1-\gamma) \times \text{neighborhood_score(child)}}
\]

//where \( \gamma \) is a predefined constant.

---

3.9 Detecting Duplicity of Web Pages

This section refers to the disappearance of Web pages from WWW and also discusses the creation, deletion and modification of Web pages. Web page change detection refers to the rate at which Web pages get updated on the WWW. The WWW contains millions of Web documents and trillions of links to
these documents (Revilla et al. 2001). The responsibility of a Web crawler does not end with the downloading and indexing of Web pages because Web pages are to be refreshed at a regular interval of time.

According to a study in (Klein and Nelson 2009), Web pages are not stable. According to (Ntoulas et al. 2004), 8% documents are created per week and 80% documents are no longer present after one year. It was also estimated that 38% contents of new document are borrowed from the existing documents. Moreover, 25% of all links are being created per week and 80% links are being replaced after a year. So, after some time, the Web pages downloaded by the crawler of a search engine become obsolete or not found on the Web. So it becomes necessary for a Web crawler to visit and update the central repository of search engine regularly. Freshness of Web pages can be defined in the following way:

Let R = (P_1, P_2, ---------, P_n) be n Web pages, then freshness of R at time t is defined as 
\[ F(R; t) = \frac{m}{n}. \]
Where m represents the number of latest Web pages and mathematically freshness of a local Web page WP may be represented as (Cho and Molina 2000) follows:

**if** (WP is latest at time t) **Then**

\[ F(P_i; t) \] is assigned a value 1 otherwise 0 is assigned to it.

Hence freshness of repository R consisting of these Web pages at time t is defined as

\[ F(R, t) = \frac{1}{n} \sum_{i=1}^{n} F(P_i; t) \]

...\(3.2\)

One other problem is the duplicity of Web documents i.e. Web containing multiple copies of the same content. The study of (Manning et al. 2009) showed that approximately 40% of the Web documents on the Web are replica of each other. So, they should not be downloaded and there must be no duplicate Web pages in central repository of the search engine.
3.10 Policies for Refreshing the Repository of the Search Engines

As the Web pages available on the Web are very dynamic in nature and they change very rapidly, it becomes necessary to keep the available Web page up to date in the central repository of the search engine. There are two popular policies for refreshing the Web pages available in the central repository of search engine.

- Incremental refreshing and
- Periodical refreshing

3.10.1 Incremental Refreshing policy

In the early days of the World Wide Web, the size of repository of search engines was very small in size. That’s why the response of Web search became available to user promptly. Similarly, the change of Web pages was also detected immediately. This process of crawling started with a set of seed URLs and the Web pages were crawled iteratively. The new URLs to be crawled were extracted from the set of downloaded Web pages. After some time, the repository can be used as a “snapshot” of the Web to build a keyword index. If a reasonable amount of seed URLs were used and were appropriately prioritized then results might be good enough to satisfy a user request. The snapshot approach works well for Web pages that are created and remain unchanged, but today, URLs are constantly being created or destroyed, and contents of URLs may change without any notice. As a result, there will always be some URLs for which the content is not present in the repository, as well as URLs whose content is different from the copy in the repository. Many new URLs can only be discovered by re-crawling old URLs whose content has now got changed. In order to minimize the impact of these inconsistencies, URLs should periodically be prioritized and revisited. The process of prioritizing and revisiting URLs is usually referred to as incremental crawling. A Web author knows that the latest Web page available in the central repository would attract the readers to visit the page again and again. Thus the more dynamic parts of the Web are those that
have the highest readership. As a result, incremental crawling strategy has become increasingly important (McCurley 2009).

**Metrics used for Incremental Crawling:** There is no proper way for measuring the inconsistency between the repository of the search engine and the actual Web. Moreover, even if there was universal agreement on the proper metric for consistency, it may be difficult to measure the ever changing Web. Three possible metrics used for this purpose are as follows:

(i) **Coverage:** The count of total number of Web pages that exist on the Web but are not present in the central repository of the search engine.

(ii) **Freshness:** The count of total number of Web pages which are present in the repository of the search engine, but whose content or size was changed after insertion into the repository.

(iii) **Age:** The average age of documents in the repository (e.g., time since they were last updated).

### 3.10.2 Periodical Refreshing Policy

With the growth of Internet almost every type of data is available on WWW. Most of the data sources are independent and are updated on regular interval of time. For example, the news available on news channels Web sites such as Cable News Network (CNN) and Aaj Tak News channel, update their contents immediately as soon as new contents are available to them. Also, many online stores update the price and availability of their products, depending on available stock and on market conditions. Since the sources are updated autonomously, the clients usually do not know exactly when and how often the sources change. However, it is believed that the clients can significantly be benefited by estimating the change frequency of the sources. But some challenges are there to get the updated information. These are as follows:

(i) **Incomplete change history:** The complete information about the Web page like how often and when the contents of a Web page or the Web page
itself get changed are not provided with the Web page. For instance, a Web
crawler can tell if a page has changed between accesses, but it cannot tell
how many times the page has been changed.

(ii) **Irregular access interval**: No one can control, how often and when a data
item is accessed from the Web cache. The access is entirely decided by the
user. So, the access interval can be arbitrary. When the limited change
history is available and the access pattern is irregular then it becomes very
difficult to estimate the change frequency.

(iii) **Difference in available information**: Different applications can provide
different levels of information for different data items. For instance, certain
Web sites provide the information about “when a page was last-modified”,
but a majority of Web sites do not provide such type of information. So,
there should be a mechanism that can avoid such type of mismatch in the
available information.

3.11 Software Agents

A software agent is a persistent, goal-oriented computer program that reacts to
its environment and runs without continuous direct supervision to perform some
function for an end user or another program. Software agents can activate and
run themselves, without any interaction with a human user. Software agents can
also initiate, oversee, and terminate other programs or agents including
applications and online intelligent agents. Software agents are being used in
areas of research (Menzer 1999) like mobile computing, information retrieval,
profiles on social networking Web sites and telecommunication etc. All software
agents are programs but not all programs are agents. For example, an e-form
(electronic form) is a computer program version of a paper form. It will
eliminate the cost of printing, storing, distributing pre-printed forms and the
wastage of obsolete forms. E-forms can be filled in faster because the
programming associated with them can automatically format, calculate, look up,
and validate information for the user. The software agents can perform the work on the behalf of its users (Mahmoud 1996; Wong et al. 1997: Sharma et al. 2005) and can communicate with system resources and other agents as required to perform their task.

3.11.1 Agent Characteristics

Several key characteristics offer minimal requirements for a software to be classified as an agent. The characteristics frequently associated with software agent are autonomy, reactivity, persistence, mobility; communication and interaction, goal-orientedness, intelligence and adaptability (Mahmoud 1996; Sharma et al. 2005).

(i) **Autonomous**: An autonomous agent is an agent program that operates in parallel with the user. Autonomy says that the agent is, conceptually at least, always running. The agent may discover a condition that might be of user interest. So, a software agent could be a program executed initially by the user which would then carry out its purpose independently without any human intervention.

(ii) **Reactivity**: It is an ability of an agent to perceive the environment and respond to changes in that environment timely (Finin and Fritzson 1994).

(iii) **Persistence**: Software agents must run or execute continuously and observe their environment regularly. The length of time for which a software agent may persist varies depending upon the task that the agent is assigned to carry out.

(iv) **Mobility**: A mobile agent should also be able to migrate from one machine to another or one network to another.

(v) **Communication and Interaction**: A mobile agent can communicate with other agents or with humans. Sometimes, this interaction plays a very important role in decision making.
(vi) **Goal Orientedness**: All the software agents either on remote machine or any other network have their own particular ambitions or goal to be achieved.

(vii) **Intelligence**: Agents also have a feature for reasoning and learning that enable an agent to pursue its goals more efficiently with less assistance from the user.

(viii) **Adaptability**: The software agents have the ability to adjust their own behavior over time by judging the internal situation or changes in the environment around them.

### 3.11.2 Classification of Software Agents

The software agents (Hector and Narasimhan 2005; Moraitakis 1997) are classified as follows:

(i) **Collaborative agents**: An agent whose implementation is focused on autonomy and cooperation with other agents in order to perform the task of its owner is called collaborative agent. The key characteristics of these agents may include autonomy and reactivity. They usually appear in multi-agent systems, and most of the time they are static. Collaborative agents are most suitable for:

- Situations where the problem is too large for a single agent and must be distributed due to resource limitations.
- Situations where there is a need for interconnection of multiple existing systems such as expert system and decision support system.
- Problems whose nature is distributed in itself like air traffic control and distributed sensor network.
- Situations in which the problem-solving capacity is distributed like health care provisions.
- Situations in which the enhancement of modularity (which reduces complexity), speed (due to parallelism) and reusability are of prime interest.
(ii) **Interface agents:** The agents focused in autonomy and learning in order to perform the tasks for their owner are called interface agents. Normally, such agents observe the actions of the user, interact with him/her and observe the action taken by the user. Interface agents are smart software that helps their owner to automate and perform their tasks.

Interface agents can assist its user in following ways:

- Learning from the user through positive or negative feedback.
- They can work as Personal Assistants.
- By taking explicit information from user.
- By taking advice (limited) from peer agents.
- Entertainment.
- Tutorial Systems.

(iii) **Mobile Agents:** The mobile agents are software processes capable of moving around wide area networks, interacting with remote hosts, collecting information on behalf of their owner and consequently returning back having to perform the duties set by its user (Moraitakis 1997; Sharma et al. 2005). A mobile agent can move to remote computer or network as shown in Figure 3.19.

The mobility feature of this agent allows it to move among different agent platforms where they can compute or perform their task. The platform from which an agent originates is referred to as home computer. One or more hosts can comprise the agent platform where different agents can interact with each other. They can cooperate or communicate with each other to set their internal state without giving all other information.

The major components of mobile agents are as follows:

- **Code:** the program written in a programming language that defines the agent’s behavior.
• **State:** the agent’s internal variables which enable it to resume its activities after moving to another host.

• **Attribute:** it describes the information about the agent. This information may include agent’s origin and owner, resource required, authentication keys and movement history etc. The agents are not able to modify these attributes by themselves.

The mobile agents are mainly used in:

• The situations where local resources CPU or storage etc. are limited. In such a situation remote resources are required to complete the task.

• Situations where asynchronously distributed computation needs to be achieved.

• Situations where the cost of bringing vast amount of remote information to the local node is huge, thus necessitating the use of a mobile agent to fetch this required information.
(iv) **Information or Internet Agents:** Information agents are tools that coordinate to perform the role of retrieval, management and manipulation of information from many distributed sources. Informational agents can be static or mobile; they can be non-cooperative or social; they may or may not be of learning nature.

(v) **Reactive Software Agents:** A reactive agent is one which can change its behavior with the changes in the environment. The major benefit of reactive agents is that they are more robust in nature and fault tolerant in complex environment than other agents. Other major benefits of using these agents are flexibility and adaptability in contrast to rigidity and slow response time of standard AI system.

### 3.12 Automatic Text Summarization

It is a process of reducing a text document with a computer program in order to create a summary that retains the most important points of the original document. As the problem of information overload has grown and the quantity of data has increased so the work on automatic text summarization has been accelerated, inspired by technology and numerous evaluation programs. For summary generation some variables such as length, writing style and syntax are taken into consideration. An example of the use of summarization technology is freesummerizer.com.

#### 3.12.1 Classification of Automatic Text Summarization

The automatic text summarizations can be classified into single document text summarization and multi document text summarization (Suneetha 2011).

Single Document Text Summarization generates the summary from a single document. Multi-document text summarization generates the summary by taking the input from multiple documents. Several Web based news clustering systems were inspired by research on multi document summarization, for example

### 3.12.2 Issues and Challenges with Automatic Text Summarizer

The issues and challenges available with the automatic text summarizers are as follows:

(i) The main issue with the automatic text summarizer is its presentation of results. They present the results as a paragraph even if the textual information needs to be present in points.

(ii) The summarization of results is based on frequency of words but some time such types of summarization results in irrelevant summarization.

### 3.13 Summary

The literatures surveys have been grouped into five categories-Web Crawler Architecture, Meta Search Engine, Automatic Text Summarization, Search Engine Evaluation and query optimization.

The Basic Crawler becomes the basis for other crawlers going to be developed. The major challenges in designing of the Basic Crawler were scalability, Information Overkill, database size and network traffic.

The Focused Crawler gathers the pages on a specific topic. The major challenges in designing of the focused crawlers were to maintain the missing relevant pages, maintain the freshness of the database, reducing the network bandwidth consumption and adverse impact on Web servers and absence of particular context.

The Parallel Crawler employs multiple crawling processes in parallel. The major challenges in designing of the parallel crawlers were multiple downloading of pages, maintaining the quality of pages and reducing the bandwidth consumption.
The Distributed Crawlers operate simultaneous crawling agents. The major challenge in designing of distributed crawlers were the distribution of URLs among different agents, maintain the priority in crawling, effective way of partitioning the collection, balancing the load of the Web servers and reducing the network bandwidth consumption.

Meta Search Engine uses multiple search engines to search the required information from the Web. The major challenges in designing of meta search engine were enhancement in search coverage, Facilitate the use of multiple search engines on a single platform, presentation of results and irrelevant results.

The Automatic Text Summarization technique is used to summarize the textual documents. The major challenge in designing the auto text summarizer is non availability of major algorithm that can provide relevant and irredundant summery of the document.

The Search Engine Evaluation used to find out the expertization of search engine in particular domain. The major challenge in designing of automated search evaluation technique was inability of evaluation on dynamic environment.