Chapter 2

SEARCH ENGINES AND PAGE RANKING ALGORITHMS

2.1 INTERNET AND WORLD WIDE WEB

The Internet is a massive network of networks [1, 2] that interconnects millions of computers wherein the information travels via a variety of protocols.

The Internet was the result of some visionary thinking by people in the early 1960s to allow interconnection between computers to share information on research and development in scientific and military fields [29]. Leonard Kleinrock of MIT developed the theory of packet switching, which formed the basis of Information transfer over the internet. Lawrence Roberts of MIT (Massachusetts Institute of Technology) connected a Massachusetts computer with a California computer in 1965 over dial-up telephone lines. It showed the feasibility of wide area networking and substantiated the Leonard Kleinrock's packet switching theory was confirmed. Lawrence Roberts moved over to DARPA (Defense Advanced Research Projects Agency) in 1966 and developed his plan for ARPANET. The Internet, then known as ARPANET, was brought online in 1969 by the Advanced Research Projects Agency (ARPA), which initially connected the major computers of four universities in the southwestern US (University of California Los Angeles, Stanford Research Institute, University of California-Santa Barbara, and the University of Utah). However, the initial Internet was used by computer experts, engineers, scientists and librarians for the purpose of information sharing within a closed society.

In 1989, Tim Berners-Lee [29] and others at the European Laboratory for Particle Physics in Switzerland (more popularly known as CERN) proposed a protocol for information distribution over the internet called HTTP (HyperText Transfer Protocol). This protocol resulted in the birth of World Wide Web (WWW) in 1991. It was based on hypertext, a system of embedding links in text to refer to another piece of text or document. Thereafter, the
concept of hypertext became the basis for the development of a standard web language called HyperText Markup Language (HTML).

The development in 1993 of the graphical browser Mosaic by Marc Andreessen and his team at the National Center for Supercomputing Applications (NCSA) was another landmark towards effective information retrieval from the WWW. Today, the Microsoft Corporation’s Internet Explorer occupies 62.7% of web browser market share [30].

The WWW [3, 31] can be precisely defined as a collection of globally distributed interlinked hypertext documents spread over the Internet. In fact, the Internet & the WWW, though used interchangeably, are not synonymous. Internet is the hardware part - it is a collection of computer networks connected through either copper wires, fiber optic cables or wireless connections whereas, the WWW can be termed as the software part – it is a collection of web pages connected through hyperlinks. Another method to differentiate between both is using the Protocol Suite i.e while internet is governed by the Internet Protocol – specifically dealing with data as whole and their physical transmission in bits, the WWW is governed by the HTTP that deals with the linking of files, documents, other resources of the WWW and transmission of information through packets.

### 2.1.1 EVOLUTION OF THE INTERNET AND WWW

As of today, internet has expanded to serve millions of users and for a multitude of purposes in all parts of the world. The internet usage statistics [32] in terms of Millions of Users from January 1995 till January 2010 is shown in Fig. 2.1. The graph depicts 18% increase in internet users in last two years.

In fact, WWW has expanded by about 2000% since its inception and is doubling in size every six to ten months [33]. Fig. 2.2 shows the growth curves of the WWW in terms of average density of a web page and average web page size. It may be observed that the average size of the web pages of the top 500 websites has more than quintupled since 2003 as it has grown from 93.7K in 2003 to over 507K in 2009 (over 5.4 times larger) [34, 35, 36]. During the same six-year period, the average density of a web page has more than doubled from 25.7 to 64.7 objects per page. Longer term statistics show that since 1995, the size of
the average web page has increased by 35 times, and the density of page has grown by about 28 times.

The number of web sites has also experienced a high growth since 1991. The WWW has experienced three growth stages [37]:

- 1998-2001: Rapid growth, at a rate of 150% per year.
- 2002-2009: Maturing growth, at a rate of 25% per year.
Netcraft’s latest Web survey [38] found approximately 234 Million Web Sites in December 2009, out of which 47 Million sites were added in only 2009. Not all of these sites are alive: some are "parked" domains, while others are abandoned web logs that haven’t been updated in ages. But even if only half the sites are maintained, there are still more than 150 Million sites that people pay to keep running.

2.1.2 WEB DOCUMENTS

A Web document [3, 8] can be defined as a file or set of related files that can be transferred from a Web server to a Web client (usually Web Browser [5]). The Web document can contain text, graphics, sound, video, or links to other documents. As hypertext is the underlying concept defining the structure of the WWW, therefore web documents are generally referred to as Hypertext Documents. The Web documents can either be static (prepared and stored in advance) or dynamic (continually changing in response to user input).

The Web employs following techniques to manage hypertext documents and other web information resources readily available to the widest possible audience:

1. A uniform naming scheme for locating resources on the Web (e.g., URIs).
2. Protocols, for access to named resources over the Web (e.g., HTTP).
3. Hypertext, for easy navigation among resources (e.g., HTML).

A brief description of above techniques is given below:

1. **Uniform Resource Identifiers (URIs)**

   Every resource available on the Web—a Web document, image, video clip or program, etc. has an address that may be encoded by a *Universal Resource Identifier (URI)* [2, 39]. A URI identifies a resource either by location or name. URIs typically consist of three pieces: the *naming scheme* of the mechanism used to access the resource, the *name of the machine* hosting the resource and the *name of the resource* itself given as a path. Consider the URI that designates the W3C home page:

   \[ http://www.w3.org/Icons/w3c_home \]
A Uniform Resource Locator (URL) [2, 39] is a specialization of URI that defines the network location of a specific representation for a given resource. Taking the same W3C example, the following URLs represent the W3C home resource:

http://www.w3.org/Icons/w3c_home.gif
http://www.w3.org/Icons/w3c_home.png

These URIs also define the file extension that indicates what content type is available at the URL. Thus, URLs form a subset of the more general URI naming scheme. URLs are used to link to another document/resource on the Web.

2. Hyper Text Transfer Protocol (HTTP)

Hyper Text Transfer Protocol (HTTP) [29], the underlying protocol of WWW, functions as a request-response protocol in the client-server computing model shown in Fig. 2.3. HTTP defines how messages are formatted and transmitted, and what actions the Web servers and clients should take in response to various commands. In HTTP, a web browser, for example, acts as a client, while an application running on a computer hosting a web site functions as a server.

![Fig. 2.3: Client-Server Model of Web Applications](image-url)
The client submits an HTTP request message to the server. The server, which stores content, or provides resources, such as HTML files and images, generates such content as required, or performs other functions on behalf of the client and returns a response message to the client. A response contains completion status information about the request and may contain any content requested by the client in its message body. The HTTP is designed to permit intermediate network elements to transmit hypertext documents/objects between clients and servers.

3. HyperText Markup language (HTML)

HTML [40] is the standard publishing language used by WWW and it provides the following features to authors of web documents:

- Publish online documents with headings, text, tables, lists, photos, etc.
- Retrieve online information via hypertext links, at the click of a button.
- Design forms for conducting transactions with remote services, for use in searching for information, making reservations, ordering products, etc.
- Include spread-sheets, video clips, sound clips, and other applications directly in the documents.

HTML offers many of the conventional publishing idioms for rich text and structured documents, but what separates it from most other markup languages is its features for hypertext and interactive documents. The basic hypertext construct is a link (or hyperlink, or Web link), which provides a connection from one Web resource to another. Although a simple concept, the link has been one of the primary forces driving the success of the Web. The link has two ends called anchors and a direction. The link starts at the "source" anchor and points to the "destination" anchor, which may be any Web resource e.g. an image, a video clip, a program, another HTML document, or an element within an HTML document, etc. In order to refer to the related documents, links are embedded into the HTML documents at suitable places. The anchor tags <a> and </a> are used for inclusion of hyperlinks as shown in the sample link given below:

\[<a \text{href}="web address">Link Text</a>\]
where the keyword "href-" sign specifies the page to which the link points. The "Web address" is any existing URL and "link text" is optional field indicating the text to be displayed for hyperlink.

The hyperlinks [41] can be of the two types as shown in Fig. 2.4. The External Links are the links which refer to the pages of other Web sites, while Internal Links refer to pages of the same Web sites. The Page Links are the intra links referring to a fragment of the same Web page and the Local Links are the links to the pages of Web site to which the page belongs. An example hyperlinked structure linking three Web pages by different types of links is shown in Fig. 2.5.

![Types of Hyperlinks in an HTML Document](image)

The Web Documents can be accessed by the community of users with the help of Information Retrieval tools. The next section describes the basic concepts of Information Retrieval and its various types.

### 2.2 INFORMATION RETRIEVAL

Information retrieval [5, 39] is fast becoming the dominant form of information access. It can be defined as:

*Information Retrieval (IR) is the task of finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers).*
In an abstract sense, IR deals with the representation, storage, organization of, and access to information items. The representation and organization of the information items should provide the user an easy access to the information in which he is interested. The general IR process is depicted in Fig. 2.6, wherein Index provides an efficient representation of the information items stored in the database. The query engine is responsible for taking user queries, retrieving the matched results/records from the index and representing them to the user in an understandable manner.

Earlier, IR used to be an activity that only a few people like reference librarians and similar professional searchers were engaged in. But, in current scenario, hundreds of millions of
people are engaged in this process in their daily routines, for example when they use a web search engine to access the Web documents on WWW or search their emails.

![Diagram of the Basic Process of Information Retrieval]

**Fig. 2.6: The Basic Process of Information Retrieval**

Often, the term Data Retrieval is used as an interchangeable term for Information Retrieval, but they are not exactly same from the IR point of view as described in the following subsection.

### 2.2.1 DATA VERSUS INFORMATION RETRIEVAL

*Data Retrieval* [42] consists mainly of determining which set of document collection contains the keywords of the user query i.e. it is mainly oriented towards finding the matched documents corresponding to queries. But, in general, these results are not enough to satisfy the user information needs. Therefore, in contrast, the user of an IR system is concerned more towards retrieving *information* about a subject than retrieving data that satisfies a given query. The data retrieval techniques aim at retrieving all objects which satisfy clearly defined conditions such as a regular expression or a relational algebra expression. Thus, for a data retrieval system, a single erroneous object among a thousand retrieved objects means total failure. For an IR system, however, the retrieved objects might be inaccurate and small errors are likely to go unnoticed. The main reason for this difference is that IR usually deals with natural language text or free text, which is not always well structured and could be semantically ambiguous also. On the other hand, a data retrieval system (such as a relational database) deals with data that has a well defined structure.
2.2.2 OFFLINE VS ONLINE INFORMATION RETRIEVAL

In *Offline* or *Traditional IR* [39, 43], data retrieval techniques are applied on a static collection of documents to retrieve the most “relevant” set of documents from corresponding to a given query. The document collection is controlled, relatively small and almost never changing, in the sense that no document was created for spamming, rather created with the intent of being selected for unrelated queries. The vector space model [43] is used for document representation, wherein documents are tokenized in words and terms are possibly filtered against a static defined stop list. Each canonical token is represented as an axis and each document as a vector in the space. If the term $t$ appears $n(t, d)$ times in document $d$, then the $t^{th}$ coordinate of $d$ is $n(t, d)$.

The traditional discipline of IR was used earlier, but in 1990’s, everything changed with the creation of the WWW. Web objects are uncontrolled collections, in the sense that billions of authors create them independently and, sometimes, they create them for spamming. In addition, Web objects represent the largest and ever changing collection of information ever created by humans. In order to adapt to this changing environment, a new discipline called *Online* or *Web Information Retrieval (WebIR)* [5, 39, 43] has been created. It uses some of the concepts of traditional IR and introduces many innovative ones. Web Search Engines have established themselves as a revolutionary working metaphor for WebIR. If someone needs information about a book, an address, a research paper, a flight ticket, or almost any other topic, they just make a query on a search engine and retrieve the desired information.

Information retrieval systems can also be distinguished by the scale at which they operate i.e. *Web search*, *Personal search* and *Enterprise search*, and it is useful to distinguish three prominent scales. In *web search*, the system has to provide search over billions of documents stored on millions of computers. Distinct issues are needed to gather documents for indexing and being able to build systems that work efficiently at this enormous scale. At the other extreme is *personal information retrieval* [43]. Example of this type of IR is integrated information retrieval in modern consumer operating systems (such as Apple’s Mac OS X Spotlight or Windows Vista’s Instant Search). Email programs usually not only provide search but also text classification. Distinct issues here include handling the broad range of
document types on a typical personal computer, and making the search system maintenance free and sufficiently lightweight in terms of startup, processing, and disk space usage that it can run on one machine without annoying its owner. In between is the space of enterprise, institutional, and domain-specific search, where retrieval might be provided for collections such as a corporation’s internal documents, a database of patents, or research articles on biochemistry. The documents are typically stored on centralized file systems and one or a handful of dedicated machines provide search over the collection. Web search is an online IR whereas other two IR systems fall into offline category.

2.2.3 PERFORMANCE MEASURES IN IR SYSTEMS

Many different measures for evaluating the performance of information retrieval systems have been proposed in the literature [5]. These measures require a collection of documents and a query. All measures described below assume a ground truth notion of relevancy i.e. every document is known to be either relevant or non-relevant to a particular query. In practice queries may be ill-posed and there may be different shades of relevancy.

1. Precision

Precision is the fraction of the relevant web documents (as per user’s information need) retrieved over total number of retrieved web documents.

\[
\text{Precision} = \frac{|\text{relevant documents} \cap \text{retrieved documents}|}{|\text{retrieved documents}|} \tag{2.1}
\]

In binary classification, precision is analogous to positive predictive value. Precision takes all retrieved documents into account. It can also be evaluated by considering only the topmost results returned by the system. This measure is also called “precision at n” or P@n.

2. Recall

Recall is the fraction of relevant web documents retrieved over all relevant web documents as analyzed by experts.
In binary classification, recall is called sensitivity. So it can be looked at as the probability that a relevant document is retrieved by the query. It is trivial to achieve recall of 100% by returning all documents in response to any query. Therefore recall alone is not enough but one needs to measure the number of non-relevant documents also, for example by computing the precision.

3. Fall-Out

Fall-out is the proportion of non-relevant documents that are retrieved out of all non-relevant documents available.

\[
\text{Fall Out} = \frac{|\text{non relevant documents} \cap \text{retrieved documents}|}{|\text{non relevant documents}|} \tag{2.3}
\]

In binary classification, fall-out is closely related to specificity. It can be looked at as the probability that a non-relevant document is retrieved by the query. It is trivial to achieve fall-out of 0% by returning zero documents in response to any query.

4. F-measure

It is the weighted harmonic mean of precision and recall. The traditional F-measure or balanced F-score is:

\[
F = \frac{2 \cdot \text{precision} \cdot \text{recall}}{(\text{precision} + \text{recall})} \tag{2.4}
\]

This is also known as the $F_1$ measure, because recall and precision are evenly weighted. The general formula for non-negative real constant $\beta$ is:

\[
F_\beta = \frac{(1 + \beta^2) \cdot (\text{precision} \cdot \text{recall})}{\beta^2 \cdot (\text{precision} + \text{recall})} \tag{2.5}
\]

where $\beta$ defines the weight of recall over precision.
Two other commonly used F measures are the $F_2$ measure, which weights recall twice as much as precision, and the $F_{0.5}$ measure, which weights precision twice as much as recall.

The next section describes the basics of Web Search Engines to understand the concept of WebIR.

### 2.3 WEB SEARCH ENGINE

The plenteous content available on the WWW is useful to millions of people worldwide. Some simply browse the Web through entry points such as *Yahoo, MSN* etc. But many information seekers use a Web search engine to begin their Web activities. A *Search Engine* [7, 8, 39] is an automated information retrieval system designed to help minimize the time required to search for desired information on the WWW. The search results are generally presented in an ordered list or in groups, and are often called hits. The results may consist of web pages, images, information and other types of files. Some commonly accessed search engines are *Google, Yahoo!, AltaVista, AOL, Bing, Clusty* etc.

Search engines can generally be categorized into three types [44]:

1. Those that are powered by robots (called *crawlers*).
2. Those that are powered by human submissions.
3. Those that work as a Meta layer on other search engines.

The three types of search engines are explained below in brief.

#### 1. Crawler Based Search Engine

These search engines use automated software agents (called crawlers) that visit a Web site, read the information on the actual site, read the site's meta tags and also follow the links that the site connects to so as to perform indexing on all linked Web sites as well. The crawler returns all that information back to a central repository, where the data is indexed. The crawler will 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. Some examples of these types of search engines are *Google, AltaVista, Lycos* etc.
2. Human powered Search Engine

Unlike crawler based search engines, this type of search engine relies on human editors to submit information that is subsequently indexed and catalogued. In fact, judgment of which pages to index does not depend on the placement of keywords in the pages, but dependent on the editors who decide whether the pages are valuable to searchers or not. The editors organize the pages in subject categories (also called directories) based on classification of subjects. Directories do not contain full text of the pages they link to; the coverage is limited based on the information submitted by editors. In present scenario, search engines like Looksmart, MSN, Excite and Yahoo rely on providers of directory data to make their search results more meaningful.

3. Meta Search Engines

The size of the Web is too large that it is almost impractical that a single search engine can index it. So, unlike the above two categories of search engines, Meta search engines do not crawl the web themselves to build their indexes, rather they create what is known as Virtual Database. On submission of a user query, they consult several other search engines for accumulating the results into a single list to be returned to the user. Metacrawler, cnet etc are the examples of Meta search engines.

Following subsections describe the general architecture of search engines and basic terminologies used by them.

2.3.1 GENERAL ARCHITECTURE OF A SEARCH ENGINE

The architecture [7, 44] of a typical crawler based search engine illustrating the Web information retrieval process is shown in Fig. 2.7. The complete process is divided into two phases:

- The back-end and
- The front-end.

At the back-end, Crawler [45, 46] is the most important component of search engine that traverses the hypertext structure of the WWW, downloads the web pages and parses them.
The parsed pages are then routed to an indexing module [47, 48] that builds the index on the basis of different terms present in the pages. The index is used to keep track of the Web pages fetched by the web crawler. The fetched pages and their hyperlinks are also used to construct a Web graph, where pages act as nodes and hyperlinks as edges. The graph is consulted by the page rank calculator to assign a link-oriented rank score (Query Independent Score/Popularity Score) [7, 49] to different web pages, which is almost independent of the user query words.

At the front-end, when user submits his query in the form of keywords on the interface of the search engine, the query processor/engine performs its execution by matching the query keywords with the document information present in the index. A page is considered as a hit if it possesses at least one of the query keywords. The matched URLs are retrieved from the
index and given to the ranking module so as to return a ranked list to the user. This module calculates the text matching scores between the user query and the matched pages i.e. the content-oriented score (Query Dependent Score/Relevance Score) [50, 132], may combine this score with the previously calculated link-oriented score and returns the final ranked pages to the query processor which in turn returns them to the user.

The current popular search engines also support a level of interaction with the user, most of which incorporate little relevance feedback beyond “find more like this” or lists containing suggested supplementary query terms also called recommendations. Following subsections are devoted to the detailed discussion of the different tasks performed by a search engine.

2.3.2 WEB CRAWLING

The primary task of a search engine is content aggregation i.e. collecting information about Web pages stored on different Web servers in its local repository and for this purpose, Web crawlers are employed. A Web crawler (also called spider, ant, automatic indexer, bot, robot or Web scutter) [45, 46, 51] is a component of search engine that browses the WWW in a methodical, automated orderly fashion, and retrieves the web pages. Crawlers traverse the sites by recursively following hyperlinks embedded in the documents. Each document encountered is stored and parsed for further collection of embedded URLs. The basic operation of a crawler is shown in Fig. 2.8. which begins with one or more URLs that constitute a Seed Set.

![Fig. 2.8: The Crawling Process of a Search Engine](image-url)
Crawlers typically maintain a *frontier*, the queue of pages which remain to be downloaded. The frontier is initialized with a seed set, specified manually. A component called *Scheduler* picks a URL from this seed set and passes it to the *Downloader* to fetch the web page at that URL. The fetched page is then parsed by a *Parser*, in order to extract links from the page, wherein each link points to another URL. The fetched pages in turn are used by the *indexer* module to build a document representation called *Index*. The extracted URLs from downloaded pages are fed to URL frontier for further downloading and so on. Crawling ceases when frontier becomes empty, or resource limit is reached. The detailed process of crawler is depicted by the algorithm given in Fig. 2.9.

```
Algorithm: Crawler()
{
  1: Step 1: Read a URL from the frontier,
  2: Determine the IP address for the host name;
  3: Download robots.txt file to determine downloading permissions;
  4: Determine the protocol of underlying host e.g. http, gopher, ftp etc;
  5: Download the document based on the protocol;
  6: Identify the document format e.g. doc, html, rtf, pdf etc.
  7: Check, whether document has already been downloaded or not;
  8: If document is fresh then extract its embedded links, else goto step 1.
  9: Add URLs to the frontier and document into repository;
}
```

Fig. 2.9: Algorithm for Crawler

Crawlers also maintain a list of detected duplicate pages (so that pages are not fetched more than once) and a scope of pages to crawl (e.g. a maximum depth, specified domain, or timeout value), both of which are checked prior to adding pages to the frontier. A *URL filter* is generally used to determine whether a URL should be excluded from the frontier based on one of several tests. For instance, the crawler may seek to exclude certain domains (say, all .com URLs) in this case the test would simply filter out the URL if it were from the .com domain. Many hosts on the Web place certain portions of their websites off-limits to crawling, under a standard known as the *robots exclusion protocol* [43]. This is done by placing a file "robots.txt" at the root of the URL hierarchy at the site. Below is given an example robots.txt file that specifies that no robot should visit any URL starting with /yoursite/temp/, except for the robot called "searchengine".

28
# robots.txt for http://www.samplesite.com
User-agent: *
Disallow: /yoursite/temp/
User-agent: searchengine
Disallow:

Once crawling is complete, the downloaded and parsed documents are indexed. The next subsection describes the index construction process.

2.3.3 DOCUMENT INDEXING

The next job of the search engine is to make an index of stored documents. The indexer distills information contained within corpus documents into a format which is amenable to quick access by the query processor. Typically, this involves extracting document features by breaking down documents into their constituent terms, extracting statistics relating to term presence within the documents and corpus, and calculating any query independent evidence. The result of this process is generally a very large "lookup table" that can provide all the URLs that point to pages where a given word occurs. The table is of course limited to the pages that are covered in the crawling process. The information stored in each entry includes the current document status, a pointer into the repository, a URL checksum, and various statistics.

Each document, which has been crawled, is assigned a docID by the indexer. There is a file which is used to convert URLs into docIDs. It consists of a list of URL checksums with their corresponding docIDs and is sorted by checksum. In order to find the docID of a particular URL, the URL's checksum is computed and a binary search is performed on the checksums file to find its docID.

2.3.3.1 Process of Indexing

In order to enhance the efficiency of retrieval of information, the index of the downloaded pages is built in advance. The major steps of the indexing process [47] are:

1. **Collection of Documents**: Collect the documents to be indexed. This collection is generally maintained by the crawler at the search engine side.
2. **Parsing of Documents**: Tokenize the text contained in the documents. For example, *friends, are, forever, and, countrymen* etc. are tokens appearing in documents.

3. **Linguistic Preprocessing of Tokens**: Perform linguistic preprocessing wherein a list of normalized tokens, called indexing terms, is produced. This step involves removing extremely common words, called *stop words* that would appear to be of little value in retrieval of relevant information. An example list of some common stop words is given in Fig. 2.10.

<table>
<thead>
<tr>
<th>a</th>
<th>an</th>
<th>and</th>
<th>are</th>
<th>as</th>
<th>at</th>
<th>be</th>
<th>by</th>
<th>for</th>
<th>from</th>
<th>has</th>
</tr>
</thead>
<tbody>
<tr>
<td>he</td>
<td>in</td>
<td>is</td>
<td>its</td>
<td>of</td>
<td>on</td>
<td>that</td>
<td>the</td>
<td>to</td>
<td>was</td>
<td></td>
</tr>
<tr>
<td>were</td>
<td>will</td>
<td>with</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Fig. 2.10**: An Example stop-word list of twenty-five semantically nonselective words

Another process, called *Stemming and Lemmatization*, is applied to reduce inflectional forms (such as *organize, organizes, and organizing*) and sometimes derivationally related forms of a word (*democracy, democratic, and democratization*) to a common base form. For instance, the following conversions can be carried out:

- am, are, is > be
- car, cars, car's, cars' > car
- analysis, analyzing, analyzer, analyzes > analy

However, *Stemming* usually refers to a crude heuristic process that chops off the ends of words in the hope of achieving this goal correctly most of the time, and often includes the removal of derivational affixes. In terms of efficiency, stemming reduces the number of unique words in the index, which in turn reduces the storage space required for the index and speeds up the search process. In terms of effectiveness, stemming improves recall by reducing all forms of the word to a stemmed form. *Lemmatization* usually refers to doing things properly with the use of a vocabulary and morphological analysis of words, normally aiming to remove inflectional endings and to return the dictionary form of a word, which is known as the *lemma.*
4. **Index the Documents**: Index the documents with respect to their term occurrences by creating an index consisting of dictionary and postings.

### 2.3.3.2 Data Structures for Index Construction

Search engines can employ various index storage methods to meet different design factors. Following are the three major index structures [53, 54] used by majority of search engines:

1. **Suffix Tree**

   A *Suffix Tree* (also called *PAT tree*) is a data structure that presents the suffixes of a given string in a way that allows for a particularly fast implementation of many important string operations. It is figuratively structured like a tree and supports linear time lookup. The tree is built by storing the suffixes of words. It supports extendable hashing, which is important for search engine indexing. A major drawback of this index is that the storage of a word in the tree may require more storage than storing the word itself. An alternate representation is a suffix array, which is considered to require less memory and supports data compression.

   The edges of the tree are labeled with strings, such that each suffix of string corresponds to exactly one path from the tree's root to a leaf. Figure 2.11 shows a Suffix tree for the string *BANANA*. Each substring is terminated with special character $\. The six paths from the root to a leaf (shown as boxes) correspond to the six suffixes *AS, NAS, ANAS, NANAS, ANANAS* and *BANANAS*. The numbers in the boxes give the start position of the corresponding suffix. Suffix links are drawn dashed.

![Fig. 2.11: Example Suffix Tree for string BANANA](image)
2. Forward Index

The forward index stores a list of words for each document. The following is a simplified form of the forward index:

<table>
<thead>
<tr>
<th>Document</th>
<th>Words</th>
</tr>
</thead>
<tbody>
<tr>
<td>Doc1</td>
<td>The, cow, gives, milk, man, says, to, eat, apple</td>
</tr>
<tr>
<td>Doc2</td>
<td>The, cat, and, the, hat</td>
</tr>
<tr>
<td>Doc3</td>
<td>The, dog, ran, away, with, the, fork</td>
</tr>
<tr>
<td>..........</td>
<td>................................</td>
</tr>
</tbody>
</table>

The rationale behind developing a forward index is that as documents are parsed, it is better to immediately store the words per document. The forward index is essentially a list of pairs consisting of a document and a word, collated by the document. The forward index is sorted to transform it to an inverted index. Converting the forward index to an inverted index is only a matter of sorting the pairs by words. In this regard, the inverted index is a word-sorted forward index as explained below.

3. Inverted Index

Many search engines incorporate an inverted index [47, 52] to quickly locate documents containing the words in a user query and then rank these documents by relevance. The name is actually redundant: an index always maps index back from terms to the parts of a document where they occur. Because the inverted index is an index of words and each word is associated with a list of its containing documents, therefore, the search engine can use direct access to find the documents associated with each word in the query in order to quickly retrieve the matching documents. The following is a simplified illustration of an inverted index:

<table>
<thead>
<tr>
<th>Document</th>
<th>Words</th>
</tr>
</thead>
<tbody>
<tr>
<td>Doc1</td>
<td>The, cow, gives, milk, man, says, to, eat, apple</td>
</tr>
<tr>
<td>Doc2</td>
<td>The, cat, and, the, hat</td>
</tr>
<tr>
<td>Doc3</td>
<td>The, dog, ran, away, with, the, fork</td>
</tr>
<tr>
<td>..........</td>
<td>................................</td>
</tr>
</tbody>
</table>
The inverted index keeps a dictionary of terms and for each term, it has a list that records which documents the term occurs in. Each item in the list is conventionally called a posting. The list is then called a postings list (or inverted list) and all the postings lists taken together are referred to as the postings as shown in Fig. 2.12.

![Fig. 2.12: The Concept of Dictionary and Postings](image)

It may be noted that the index can only determine whether a word exists within a particular document and it stores no information regarding the frequency and position of the word, therefore, such an index does not rank matched documents. In some designs, index includes additional information such as the frequency of words in each document or the positions of a word in each document, which enables search algorithms to identify word proximity to support searching for phrases and ranking the relevance of documents to the query.

### 2.3.3.3 Indexing Techniques

As Inverted Index is the data structure employed by popular search engines, therefore many efficient indexing techniques have been proposed in the literature for inverted index construction. Below are given some important indexing techniques [43]:

1. **Block Sort based Indexing**

   In Block Sort-based Indexing (BSBI), documents are parsed into <termID, docID> pairs and accumulated in memory until a block of a fixed size is full. The block is then inverted and written to disk. The Inversion Process involves two steps: sort the <termID, docID> pairs and collect all pairs with the same termID into a postings list, where a posting is simply a
docID. The result, an inverted index for the block which is just read, is then written to disk. Finally, the technique simultaneously merges multiple blocks into one large merged index as shown in Fig. 2.13. BSBI has excellent scaling properties, but it needs a data structure for mapping terms to termIDs. For very large collections, this data structure does not fit into memory.

![Fig. 2.13: Merging in Block Sort based Indexing](image)

2. Single-Pass In-Memory Indexing

A more scalable alternative is Single-Pass In-Memory Indexing (SPIMI). This technique uses terms instead of termIDs, writes each block’s dictionary to disk, and then starts a new dictionary for the next block. SPIMI can index collections of any size as long as there is enough disk space available.

In this technique, terms/tokens are processed one by one and when a term occurs for the first time, it is added to the dictionary and a new postings list is created. The difference between BSBI and SPIMI is that SPIMI adds a posting directly to its postings list. Instead of first collecting all <termID, docID> pairs and then sorting them as in BSBI, each postings list is dynamic and is immediately available to collect. This has two advantages: It is faster because no sorting operation is required and also saves memory, the reason being that it keeps track as to which posting a term belongs to. Therefore, the termIDs of postings need not be stored resulting in much larger blocks, the index construction process also becomes efficient.
Web collections are often so large and dynamic that index construction can not be performed efficiently on a single machine. Therefore, distributed and dynamic indexing algorithms are generally applied for building the effective index. Recent researches are also focused towards semantic and context-driven indexes so as to better understand the user queries.

2.3.4 QUERY PROCESSING

User interacts with the Web by submitting queries on the interface of search engine. A web search query is a query that a user submits so as to satisfy his/her information needs. Web search queries are distinctive in that they are unstructured and often ambiguous; they vary greatly from standard query languages which are governed by strict syntax rules. There are four broad categories [55] that cover most web search queries:

1. Informational queries: Queries that cover a broad topic (e.g., colorado or trucks) for which there may be thousands of relevant results.
2. Navigational queries: Queries that seek a single website or web page of a single entity (e.g., youtube or delta airlines).
3. Transactional queries: Queries that reflect the intent of the user to perform a particular action, like purchasing a car or downloading a screen saver.
4. Connectivity queries: Queries that report on the connectivity of the indexed web graph (e.g., Which links point to this URL?, and How many pages are indexed from this domain name?).

Most of the search engines also support Boolean operators (such as OR, AND, NOT) and parentheses to facilitate the user in restricting his search to the desired content. A user who is looking for documents that cover several topics (facets) can describe his query as a disjunction of characteristic words, such as “vehicles OR cars OR automobiles”. A faceted query is a conjunction of such facets; e.g. a query such as “(electronic OR computerized) AND (voting OR elections OR election OR balloting OR electoral)” is likely to find documents about electronic voting even if they omit one of the words "electronic" and "voting", or even both.

The submitted queries are handled by the query processor [39, 44] of search engine for their execution. The query is parsed to generate terms that are searched in the index to find the
matched documents. These matched documents are generally ranked by using page ranking mechanisms and returned to the user. Various steps involved in query processing are parsing, linguistic preprocessing, query representation, matching and ranking. The query evaluation algorithm of a typical query processor is shown in Fig. 2.14.

Algorithm: Query Evaluation \( (q) \)

Input: User query \( q \)
Output: A set of URLs (or documents).

1. Step 1: Parse the query to generate terms.
2. Apply Spell checking, stop word removal, stemming and Lemmatization to create an optimized representation of query terms.
3. Search terms in the dictionary and retrieve corresponding postings from the index.
4. Apply intersection or union operators according to the query on the postings retrieved in step 3.
5. Compute the rank of the documents retrieved by applying step 4.
6. Sort the documents according to their ranks and return.

Fig. 2.14: Query Evaluation in a Typical Search Engine

2.3.5 PAGE RANKING

As described in the last section, Page Ranking [132] methods are generally applied by the search engines to present the result pages in an ordered manner to the user. These methods generally use Web Mining Techniques to find out the rank scores. Some algorithms rely only on the link structure of the pages to find out their popularity scores (using Web structure mining), some use content inside the pages to return their relevance scores (using Web content mining), while others use combined measures i.e. links as well as content of the page to assign a rank score to the concerned page. A number of page ranking mechanisms have been proposed in the literature such as PageRank [7], HITS [26], Weighted PageRank [58], Page Content Rank [50], SALSA [56, 57] etc.

Some prevalent page ranking algorithms have been discussed in next few subsections.

2.3.5.1 PageRank Algorithm

Brin and Larry Page [7, 24] developed a ranking algorithm used by Google, named PageRank \( (PR) \) after Larry Page (cofounder of Google search engine), that uses the link
structure of the web to determine the importance of web pages. Google uses $PR$ to order its search results so that the result pages that seem more important move up in the results of a search accordingly. This algorithm states that if a page has some important incoming links to it then its outgoing links to other pages also become important. Therefore, $PR$ takes incoming links into account and propagates the rank scores through outgoing links. Thus, a page obtains a high rank if the sum of the ranks of its incoming links is high.

The $PR$ algorithm considers more than 25 billion web pages on the WWW to assign them the rank scores. When some query is given, Google combines precomputed PageRank scores with text matching scores to obtain an overall ranking score for each resulted web page in response to the query. Although many factors are considered while determining the overall rank but $PR$ algorithm is the heart of Google.

A simplified version [5] of PageRank is defined in (2.6):

$$PR(u) = c \sum_{v \in B(u)} \frac{PR(v)}{N_v}$$

where $u$ represents a web page, $B(u)$ is the set of pages that point to $u$, $PR(u)$ and $PR(v)$ are rank scores of page $u$ and $v$ respectively, $N_v$ denotes the number of outgoing links of page $v$. $c$ is a factor used for normalization.

In $PR$, rank score of a page (say $p$) is equally divided among its outgoing links. The values assigned to the outgoing links of page $p$ are in turn used to calculate the ranks of the pages pointed to by $p$. An example showing the distribution of page ranks is illustrated in Fig. 2.15.

![Fig. 2.15: Distribution of Page Ranks](https://via.placeholder.com/150)
Later on, PageRank was modified according to the random-surfer model, observing that not all users follow the direct links on WWW. The modified version is represented by expression (2.7).

\[
PR(u) = (1 - d) + d \sum_{v \in B(u)} \frac{PR(v)}{N_v}
\]

(2.7)

where \(d\) is a damping factor which represents the probability of users’ following the direct links and \((1 - d)\) is the page rank distribution from non-directly linked pages. It is usually set to 0.85. The following example illustrates the working of PR:

**Example 2.1**

Consider the hyperlinked structure shown in Fig. 2.16, where A, B and C are three web pages. The PageRanks for pages A, B and C can be computed by using (2.7):

![Diagram of hyperlinked structure]

Fig. 2.16: Example Hyperlinked Structure

\[
\begin{align*}
PR(A) &= (1-d) \cdot d((PR(B)/2 + PR(C)/2) \\
PR(B) &= (1-d) \cdot d( PR(A)/1 + PR(C)/2) \\
PR(C) &= (1-d) \cdot d(PR(B)/2)
\end{align*}
\]

(2.7a) (2.7b) (2.7c)

By calculating the above equations with \(d=0.5\) (say), the page ranks of pages A, B and C become:

\[PR(A) = 1.2, \ PR(B) = 1.2, \ PR(C) = 0.8\]

In order to determine page rank values for a small set of pages, the equation method described above can be easily used but the web consists of billions of documents and it is not possible to find a solution by this method. Thus, iterative method is generally employed wherein, each page is assigned a starting page rank value of 1 as shown in Table 2.3. These
rank values are iteratively substituted in page rank equations to find the final values. In
general, many iterations could be followed to normalize the page ranks.

<table>
<thead>
<tr>
<th>Iteration</th>
<th>PR(A)</th>
<th>PR(B)</th>
<th>PR(C)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1.0</td>
<td>1.25</td>
<td>0.75</td>
</tr>
<tr>
<td>2</td>
<td>1.0</td>
<td>1.188</td>
<td>0.813</td>
</tr>
<tr>
<td>3</td>
<td>1.0</td>
<td>1.203</td>
<td>0.797</td>
</tr>
<tr>
<td>4</td>
<td>1.0</td>
<td>1.199</td>
<td>0.801</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

It may be noted that in this particular example, $PR(B) > PR(A) > PR(C)$.

2.3.5.2 Weighted PageRank Algorithm

Wenpu Xing and Ali Ghorbani [58] proposed an extension to standard PageRank called
Weighted PageRank (WPR). It assumes that if a page has more incoming and outgoing links
it becomes more popular. This algorithm does not divide the rank value of a page evenly
among its outgoing linked pages; rather it assigns larger rank values to more popular pages.
Each outgoing link gets a value proportional to its popularity or importance. The popularity
of a page is assigned in terms of weight values corresponding to incoming and outgoing links
and are denoted as $W^{in}(v, u)$ and $W^{out}(v, u)$ respectively. $W^{in}(v, u)$ (2.8) is the weight of link(v, u)
calculated based on the number of incoming links of page $u$ and the number of incoming
links of all reference (outgoing linked) pages of page $v$.

$$W^{in}_{(v, u)} = \frac{I_u}{\sum_{p \in R(v)} I_p}$$  \hspace{1cm} (2.8)

where $I_u$ and $I_p$ represent the number of incoming links of page $u$ and page $p$, respectively.
$R(v)$ denotes the reference page list of page $v$.

$W^{out}(v, u)$ is the weight of link(v, u) calculated based on the number of outgoing links of page
$u$ and the number of outgoing links of all reference pages of page $v$ as given in (2.9).
where $O_u$ and $O_p$ represent the number of outgoing links of page $u$ and page $p$, respectively. Considering the importance of pages, the original PageRank formula (2.7) is modified as given in (2.10).

$$WPR(u) = (1-d) + d \sum_{v \in \delta(u)} WPR(v)W_{in}^{u,v}W_{out}^{v,u}$$

The following example illustrates the working of WPR.

**Example 2.2**

To illustrate the working of WPR refer again to Fig. 2.16. The rank equations become:

$$WPR(A) = (1-d) + d(WPR(B)W_{in}^{A,B}W_{out}^{B,A} + WPR(C)W_{in}^{A,C}W_{out}^{C,A})$$

$$WPR(B) = (1-d) + d(WPR(A)W_{in}^{B,A}W_{out}^{A,B} + WPR(C)W_{in}^{B,C}W_{out}^{C,B})$$

$$WPR(C) = (1-d) + d(WPR(B)W_{in}^{C,B}W_{out}^{B,C})$$

The weights of incoming as well as outgoing links can be calculated as:

$$W_{in}^{B,A} = 1/2$$ and $$W_{out}^{C,A} = 1/3$$

$$W_{in}^{A,B} = 1$$ and $$W_{out}^{A,B} = 1$$

$$W_{in}^{C,B} = 1/2$$ and $$W_{out}^{C,B} = 2/3$$

$$W_{in}^{B,C} = 1/3$$ and $$W_{out}^{B,C} = 2/3$$

After substituting $d=0.5$ and above calculated weight values, page ranks of pages A, B and C get the values as given below:

$$WPR(A) \approx 0.65, WPR(B) \approx 0.93, WPR(C) \approx 0.60$$

Here $WPR(B) > WPR(A) > WPR(C)$. It may be noted that the page ranks obtained by PR and WPR are different.
2.3.5.3 Page Content Rank Algorithm

Jaroslav Pokorny and Jozef Smizansky [50] gave a ranking method for calculating the content/relevance score of the pages, called Page Content Rank (PCR). This method combines a number of heuristics that seem important for analyzing the content of web pages e.g. Term frequency (TF), position of terms, synonym classes, document frequency (DF) etc. The page relevance is determined on the basis of importance of terms contained in the page; while the importance of a term is specified with respect to the submitted query by using above heuristics.

PCR asserts that the importance of a page \( P \) is proportional to the importance of all terms (normalized tokens) in \( P \). This algorithm uses the usual aggregation functions like Sum, Min, Max, Average, Count and the function called Sec moment given in (2.11) particularly for calculating the importance of a page.

\[
\text{Sec\_moment}(S) = \frac{\sum_{i=1}^{n} x_i^2}{n}
\]  

(2.11)

where \( S = \{x_i | i=1..n\} \), \( n = |S| \) and \( x_i \) is the importance of a term. The set \( S \) is constructed for every page that stores the importance of terms contained in the page. Sec moment is primarily used by PCR as it increases the influence of extreme values in the result in contrast to Average function.

On submission of a user query \( (q) \) to a generic search engine (like Google), PCR retrieves the set of returned ranked pages \( (R_q) \) and extracts a set of top \( n \) ranked pages \( (R_{q,n}) \) from \( R_q \). The importance of every page (calculated by 2.11) in \( R_{q,n} \) imparts a new order to the pages, which truly represents the pages according to their content scores in opposition to the PR and WPR.

In general, the working of PCR method can be described in the following four steps:

1. **Term extraction:** A parser extracts terms from each page present in \( R_q \). An inverted list is built in this step which is used in step (ii).

2. **Parameter Calculation:** Statistical parameters such as a Term Frequency and occurrence positions; as well as linguistic parameters such as frequency of terms in the natural language are calculated and synonym classes are identified.
3. **Term classification:** Based on parameter calculations in step (ii), the importance of each term is determined. For this purpose, a neural network is used as a classifier that is learnt on a training set of terms. Each parameter corresponds to excitation of one neuron in the input level and the importance of a term is given by excitation of the output neuron in the time of termination of propagation.

4. **Relevance Calculation:** Page relevance scores are determined on the basis of importance of terms present in the page, which have been calculated in step (iii). The new score of a page $P$ is equal to the average importance (2.11) of terms in $P$.

The main disadvantage of PCR is its high time complexity. Moreover, it requires a training set of terms, a database of terms in natural language and information about synonym classes for calculating the importance of terms, which further increase its preprocessing time. The method relies totally on the content of the pages, ignoring their hyperlinked information. The next section describes a very prevalent algorithm, HITS that came very next to PR algorithm.

### 2.3.5.4 HITS Algorithm

Kleinberg [26] developed an algorithm called *Hyperlink-Induced Topic Search (HITS)* that assumes that for every user query, there always is a set of authority pages that are relevant, popular and focus particularly on the query. It also identifies a set of hub pages that contain useful links to relevant pages/sites and also links to many authorities as shown in Fig. 2.17. HITS assume that if the author of page $p$ provides a link to page $q$, then $p$ confers some authority on page $q$.

![Fig. 2.17: Hubs and Authorities](image-url)
In general, a search engine may not retrieve all relevant pages for a user query; therefore, initial pages returned by a search engine are considered by HITS as a good starting point to move further. But relying only on the initial pages does not guarantee that authority and hub pages are also retrieved efficiently and therefore, HITS uses the following method to find the relevant pages regarding the user query. The method works in two major steps:

**Step 1 - Sampling Step**

In this step, a set of relevant pages for the given query is collected i.e. a sub graph of web graph is retrieved which is rich in authority pages. The algorithm starts with a root set $R$ (approx. 200-300 pages) selected from the result list of a usual search engine. Starting with $R$, a set $S$ containing most of the strongest authorities is obtained keeping in mind that $S$ is relatively small and rich in relevant pages. HITS algorithm expands the root set $R$ into a base set $S$ by using the algorithm given in Fig. 2.18.

**Algorithm: Base_Set_Determination()**

```plaintext
Input: Root set $R$
Output: Base set $S$

{ 
    Step 1: Initially set $S=R$;
    2: For each page $p \in S$, do Steps 3 to 5;
    3: Let $T$ be the set of all pages $p (\in S)$ points to;
    4: Let $F$ be the set of all pages that point to $p$;
    5: Let $S = S + T +$ some or all of $F$;
    6: Delete all links with the same domain name;
    7: Return $S$;
}
```

Fig. 2.18 Algorithm to determine Base Set

**Step 2 - Iterative Step: Finding Hubs and Authorities**

This step finds hubs and authorities using the output of sampling step. The algorithm for finding hubs and authorities is shown in Fig. 2.19.

Fig. 2.20 shows how to calculate the authority and hub scores. Hubs and authorities are assigned relative weights wherein an authority pointed to by several highly scored hubs is considered a strong authority, while a hub that points to several highly scored authorities is considered to be a popular hub.
Algorithm: Hub Authority Determination()

Input: Base set \( S \)
Output: A set of Hubs and a set of Authorities

1. Let a page \( p \) has a non-negative authority weight \( x_p \) and hub weight \( y_p \). Pages with relatively large weights \( x_p \) will be classified as authorities, similarly pages with large weights \( y_p \) as hubs:
2. The weights are normalized so that squared sum for each type of weight is 1.
3. For a page \( p \), the value of \( x_p \) is updated to be the sum of \( y_q \) over all pages \( q \) linking to \( p \). 
4. The value of \( y_p \) is updated to be the sum of \( x_q \) over all pages \( q \) linked to by \( p \).
5. Continue with step 2 unless a termination condition has been reached.
6. Output the set of pages with the largest \( x_p \) weights i.e. authorities and those with the largest \( y_p \) weights i.e. hubs.

Fig. 2.19 Algorithm to determine Hubs and Authorities

\[ x_p = \sum_{q \in B(p)} y_q \]  \hspace{1cm} \[ y_p = \sum_{q \in R(p)} x_q \]

Fig. 2.20 An example of HITS operation

If \( B(p) \) and \( R(p) \) denote the set of referrer and reference pages of page \( p \), respectively. The scores of hubs and authorities are calculated as follows:

\[ x_p = \sum_{q \in B(p)} y_q \]  \hspace{1cm} \[ y_p = \sum_{q \in R(p)} x_q \]

HITS algorithm suffers with some limitations as described below:

- *No clear distinction between hubs and authorities:* A well defined distinction between hubs and authorities is not carried out since many sites act as hubs as well as authorities.

44
• *Topic drift:* If there is a tightly connected arrangement of documents on the web, they may accumulate in the results of HITS, while they may not be most relevant to the user query in some instances.

• *Automatically generated links:* Some links are automatically generated and represent no human judgment, but HITS gives them equal importance.

• *Non-relevant documents:* Some queries can return non-relevant documents in the result list and it can lead to erroneous results, as root set will not be appropriate.

• *Efficiency:* The performance of the algorithm is not good in the real time.

A number of proposals [59, 60] like Probabilistic HITS, Weighted HITS etc. have been proposed in the literature for modifying HITS. A brief comparison of the algorithms described above is given in the next section.

**2.3.5.5 Comparison Study of Page Ranking Algorithms**

A critical look at the available literature indicates that almost every algorithm suffers with a couple of limitations. HITS, like PageRank and WPR, is an iterative algorithm based on the link structure of the documents; however it does have some major differences: it is executed at query time, not at indexing time; it calculates two scores per document as opposed to a single score; it is processed on a small subset of relevant documents, not all documents. PR and WPR differentiate themselves with PCR and HITS as they mainly focus on the hyperlink structure of the pages instead of their contents.

PCR, like HITS, also chooses a subset of documents from the result list for further processing but applies a classification scheme which is not performed by HITS. Thus, it can be concluded that all page ranking algorithms possess differences [132] in their modes of working, which is briefly outlined in Table 2.4.

The next section describes an introduction to the logs (also called *Query Logs*) maintained by Search Engines for storing the user interactions with the web.
### Table 2.4 Comparison of Page Ranking Algorithms

<table>
<thead>
<tr>
<th>Algorithm → Parameters ↓ Main Technique Used</th>
<th>PageRank</th>
<th>Weighted PageRank</th>
<th>Page Content Rank</th>
<th>HITS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Computes scores at indexing time not at query time. Results are sorted according to importance of pages.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>I/P Parameters</strong></td>
<td>Backlinks</td>
<td>Backlinks, forward links</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Working levels</strong></td>
<td>(N^*)</td>
<td>(&lt; O(\log N))</td>
<td>(O(m^*))</td>
<td>(&lt; O(\log N)) (higher than WPR)</td>
</tr>
<tr>
<td><strong>Complexity</strong></td>
<td>(O(\log N))</td>
<td>(&lt; O(\log N))</td>
<td>(O(m^*))</td>
<td>(&lt; O(\log N)) (higher than WPR)</td>
</tr>
<tr>
<td><strong>Relevancy</strong></td>
<td>Less</td>
<td>Less (higher than PR)</td>
<td>More</td>
<td>More (less than PCR)</td>
</tr>
<tr>
<td><strong>Importance</strong></td>
<td>More</td>
<td>More Higher than PR</td>
<td>less</td>
<td>less</td>
</tr>
<tr>
<td><strong>Quality of result</strong></td>
<td>Medium</td>
<td>Relevancy is ignored. Method computes scores at a single level.</td>
<td>Approx equal to WPR</td>
<td>Less than PR</td>
</tr>
<tr>
<td><strong>Limitations</strong></td>
<td></td>
<td></td>
<td>Importance of pages is totally ignored.</td>
<td>Topic drift and efficiency problems</td>
</tr>
</tbody>
</table>

\(^*n\): number of pages chosen by the algorithm, \(N\): number of web pages, \(m\): Total number of occurrences of query terms in \(n\) pages

### 2.3.6 SEARCH ENGINE QUERY LOGS

The query logs [61, 62] constructed by the search engines act as a good resource for recording users’ search histories and the necessary information about users’ browsing behavior over the search results. An entry in the log records every single access made by users corresponding to their search queries. Thus, a log mainly contains users’ queries and corresponding visited URLs, as well as other information about their browsing activities. The click-through and navigation patterns stored in the logs can capture derivative traces, which can further be utilized to characterize the users and their interests.
A typical query log [62] can be regarded as a file consisting of a series of requests, wherein a request consists of a number of fields. The important fields are outlined below along with their description:

- **UserID**: An anonymous user ID/number that corresponds to a real search engine user.
- **Query**: The query issued by the user, which is case shifted with most punctuations removed.
- **QueryTime**: The time at which the query was submitted to the search engine.
- **ItemRank**: If the user clicked on a search result, the rank of the item on which he clicked is listed.
- **ClickURL**: If the user clicked on a search result, the domain portion of the URL in the clicked result is listed.

A sample query log segment of anonymous users in AOL query log data is shown in Table 2.5 to have an understanding of various fields.

<table>
<thead>
<tr>
<th>AnonID</th>
<th>Query</th>
<th>QueryTime</th>
<th>ItemRank</th>
<th>ClickURL</th>
</tr>
</thead>
<tbody>
<tr>
<td>100218</td>
<td>YMCA</td>
<td>3/1/2010 11:08</td>
<td>1</td>
<td><a href="http://www.ymcaie.ac.in">www.ymcaie.ac.in</a></td>
</tr>
<tr>
<td>100218</td>
<td>YMCA</td>
<td>3/1/2010 11:08</td>
<td>2</td>
<td><a href="http://www.ymcaie.ac.in">www.ymcaie.ac.in</a></td>
</tr>
<tr>
<td>100218</td>
<td>YMCAIE Faridabad</td>
<td>3/1/2010 11:12</td>
<td>2</td>
<td><a href="http://www.ymcaie.ac.in">www.ymcaie.ac.in</a></td>
</tr>
<tr>
<td>100218</td>
<td>Engineering College</td>
<td>3/1/2010 11:13</td>
<td>4</td>
<td><a href="http://www.edutech.ac.in">www.edutech.ac.in</a></td>
</tr>
<tr>
<td>100219</td>
<td>Engineering Institutes</td>
<td>3/1/2010 11:23</td>
<td>2</td>
<td><a href="http://www.linkedin.com">www.linkedin.com</a></td>
</tr>
<tr>
<td>100219</td>
<td>journals</td>
<td>3/1/2010 11:24</td>
<td>3</td>
<td><a href="http://www.journals.com">www.journals.com</a></td>
</tr>
</tbody>
</table>

Most commercial web search engines do not disclose their search logs, so information about what users are searching for on the Web is difficult to obtain. Nevertheless, a study in 2001 [63] analyzed the queries from the *Excite* search engine that showed some interesting characteristics of web search:

- The average length of a search query was 2.4 terms.
- About half of the users entered a single query while a little less than one third of users entered three or more unique queries.
- Close to half of the users examined only the first one or two pages of results (10 results per page).
• Less than 5% of users used advanced search features (e.g., Boolean operators).
• If the first page of results does not satisfy the users, they rather submit another query than checking the second page.

A 2005 study of Yahoo's query logs revealed 33% of the queries from the same user were repeat queries and that 87% of the time the user would click on the same result [64]. This suggests that many users use repeat queries to revisit or re-find information.

In addition, much research has shown that a small portion (approx 20%) of the terms observed in a large query log (e.g. > 100 million queries) is used most often, while the remaining terms are used less often [65]. This 80-20 rule allows search engines to employ optimization techniques such as index or database partitioning, caching and pre-fetching.

The information contained in query logs has been used in many different ways, for example to provide context during search, to classify queries, to infer search intent, to facilitate personalization and to uncover different aspects of a topic etc. In various studies, researchers and search engine operators have used information from query logs to learn about the search process and to improve search engine’s efficiency. Besides this, query logs are also being used to infer semantic concepts or relations from the users’ queries to know their inclinations and general trends for different query terms. Below is specified some research done in this area:

• **Query Refinement and Expansion:** Query expansion has become an essential information retrieval approach that interactively recommends new terms related to a particular query. As general keyword based queries are likely to miss important results due to unstructured and semantically heterogeneous nature of the Web, therefore query expansion is considered an effective method to bridge the gap between users’ internal information needs and external query expressions. Thesaurus-based query expansion [66, 68] generally relies on statistics to find certain correlations between query terms. Cui et al. [67] has mined click-through records of search results from query logs to establish mappings from query terms to strongly correlated document terms which are then used for query expansion.
**Infer Semantic Contexts:** In [69], a novel method, Q-Rank, has been proposed to leverage the implicit feedbacks from the logs about users’ search intents. The authors claimed that to improve the relevance ranking for underspecified queries, a better understanding of users’ search goals is required. By analyzing the semantic query context extracted from the query logs, they invented Q-Rank to effectively improve the ranking of search results for a given query. Experiments showed that Q-Rank outperforms the current ranking system of large-scale commercial Web search engines, improving the relevance for 82% of the queries with an average increase of 8.99% in terms of discounted cumulative gains. Because Q-Rank is independent of the underlying ranking algorithm, it can be integrated with existing search engines.

**Identify Query Classification Terms:** Classification of search queries is a complex and computationally challenging task. Typically, search queries are short, reveal very few features per single query and are therefore a weak source for traditional machine learning. Isak Taska et al. [62] presented a method that combines limited manual labeling, computational linguistics and information retrieval to classify a large collection of web search queries. A short set of manually chosen terms that are known a priori to be of interest to a particular class is used to pick a small number of actual queries from a commercial search engine log. These queries are then submitted to a commercial search engine and the returned search results are used to find more class related terms. The authors claimed that up to 48% of the unlabeled set can be classified using this method.

**Infer Semantic Relations:** Ricardo Baeza-Yates et al. [70] carried out a study on a large query log of more than twenty million queries with the goal of extracting the semantic relations. They proposed a novel way to represent queries in a vector space [71] based on query-click bipartite graph. A sample bipartite graph is shown in Fig. 2.21, where one set of vertices represent the user submitted queries, while the other set represents the corresponding clicked URLs. Authors analyzed that query-click graph produced by sample log appears less sparse than query-based graphs (shown in Fig. 2.22 and 2.23) constructed by previous researches [72] and represent distribution of topics that people want in the Web. This representation allows drawing interesting semantic relationships between queries.
Caching and Prefetching: Caching and prefetching are well known strategies for improving the performance of search engines. The heart of a caching system is its page replacement policy, which selects the pages to be replaced in a proxy cache when a request arrives. By the same token, the essence of a prefetching algorithm lies in its ability to accurately predict future requests. In [16], server side prefetching is proposed in order to improve Web latency. Of course, this technique can be useful if pre-fetched object or page appears to be the target of next request of the user. So, prefetching must be subordinated to prediction of user browsing activities from query.
logs. In [74] time-series analysis and digital signal processing are used to model web prefetching and caching systems. Ronney Lampel et al. [73] proposed the concept of PDC (Probability Driven Cache). The idea is to associate a probability distribution to all the possible queries that can be submitted to a search engine. The distribution is built over the statistics computed on the previously submitted queries. In practice, the higher the probability of a query to be submitted the higher it is ranked within the cache once it is actually appeared. Furthermore, authors also examined the prefetching of search results and found that by integrating prefetching into PDC, a hit ratio of 0.53 can be attained. Unfortunately the policy seems to be quite expensive in terms of time-to-serve for each request (in particular those causing a cache miss).

All of the above specified research involves in a way or another, the application of web mining techniques. But before discussing the concepts of Web Mining in the next chapter, some of the major issues pertaining to the design of effective and efficient search engines have been described in the next section.

2.3.7 DESIGN ISSUES IN SEARCH ENGINES

Although, the current search engines come up with state of art techniques [44] and are more efficient than the traditional search engines, there are still many issues which need to be addressed as described below:

1. **Large Volume:** Today's web consists of billions of documents, the extraction of desired content from which is a tedious task. Search engines should be able to index most of the information available on web in an efficient manner.

2. **Distributed nature of Data:** The information on the web is distributed across various servers employing different platforms. Search engine must be designed in a way to cope up with this distribution and accumulate the content in its local repository.

3. **Redundant Data:** One of the essential features, which led to the explosive growth of the web, is uploading of multiple copies of web documents on the WWW. The tremendous volume of such identical or near identical web documents poses challenges to the performance and scalability of web search engines.
4. **Dynamic Data:** The web site (or document) developers continuously update their sites (or documents) to incorporate the fresh content so as to meet user requirements. A search engine should be aware of these changes and return the current content.

5. **Heterogeneous nature of Documents:** With time, new Web technologies like Hidden web, Semantic web, Web 2.0 etc. are envisioned as the next generation of WWW to help address various complications with the traditional web. Improved technology has also facilitated an increase in the number of file formats supported by web documents posing a challenge towards the performance of present day search engines.

6. **Extensibility:** Search engines should be extensible in the sense so as to support third party functional modules e.g. mining modules, caching and prefetching modules, ranking modules and query expansion modules etc. to make them more efficient.

7. **Relevancy of Results:** As most of the search engines are keyword based, the retrieval of relevant web documents is a challenging task. Search engines generally return so many search results that user wastes most of the time sifting between them for uncovering the desired information, thus leading to the problem of Information Overkill. Search engines must be capable of returning desired documents at least on the top of result list.

In order to resolve most of the above said challenges, Web Mining plays an important role. The next chapter is devoted to the survey on web mining, its categories and its applications in the context of search engines.