1

INTRODUCTION TO INFORMATION RETRIEVAL

Unit Structure

1.0 OBJECTIVES

- To define information retrieval
- Understand the importance and need of information retrieval system
- Explain the concept of subject approach to information
- Illustrate the process of information retrieval

1.1 INTRODUCTION AND HISTORY OF IR

In his influential 1968 textbook, information retrieval pioneer Gerard Salton (Salton, 1968), who was a key figure from the 1960s until the 1990s, put out the following definition:

“Information retrieval is a field concerned with the structure, analysis, organization, storage, searching, and retrieval of information”.

This concept is still relevant and correct today despite the enormous advancements in search technology and understanding over the previous 40 years. Since "information" is such a broad concept, information retrieval encompasses study on a wide range of informational domains and applications.

Since the 1950s, text and text documents have been the field's main focus. Documents come in a wide variety of forms, including web pages, emails, books, academic papers, and news articles to name just a few. Every one of these documents has some structure to it, including the title, author, date, and abstract details related to the content of articles published in scholarly publications.When referring to database records, the components of this structure are referred to as attributes or fields. The key distinction between a document and a normal database record, like one for a bank account or a ticket reservation, is that a document's content is mostly presented as text, which is a rather unstructured format.

Consider the details present in the account number and current amount, two typical account record attributes, to demonstrate this distinction. Both have very clearly defined formats (a six-digit integer for an account number, for instance, and a real number with two decimal places for balance), as well as meanings.As a result, it is simple to develop algorithms to find the records that respond to queries like "Find account number 321456" or "Find accounts with balances larger than $50,000.00." It is very easy to compare the values of these attributes.

Think about a recent news report about a bank merger. The headline and the story's source are some of the story's qualities, but the story's actual content is what matters most. This important piece of data would normally be recorded in a database system as a single huge attribute with no internal structure. The majority of searches for this topic on search engines like Google will be of the type "bank merger" or "bank takeover." In order to conduct this search, we must create algorithms that can evaluate if the tale contains the information sought by comparing the text of the query with the text of the story.Providing a definition for a term, sentence, paragraph, or piece of news

Comparing literature is challenging since describing a tale is harder than defining an account number. The foundation of information retrieval is the understanding and modelling of how individuals compare texts, as well as the development of computer algorithms that efficiently carry out this comparison.Information retrieval applications increasingly use multimedia documents having structure, substantial text content, and other media. Pictures, video, and audio, including voice and music, are common forms of information media.Scanned document images are crucial in some applications, such legal support. Similar to text, the content of these mediums is challenging to define and contrast. Instead of using the contents themselves, present technology for searching non-text materials relies on text descriptions of their contents, but advancements are being made in methods for direct comparison of photographs, for instance.

Information retrieval encompasses a variety of tasks and applications in addition to a variety of media. In the typical search situation, a user types a query into a search engine and receives results in the form of a ranked list of documents. Search is an essential component of applications in businesses, the government, and many other fields, even though web search is by far the most popular application involving information

retrieval. A specific type of web search called vertical search limits the domain of the search to a single subject. Enterprise search is locating the necessary information among the enormous variety of digital assets dispersed throughout a company intranet.The majority of the information will be found in sources like emails, reports, presentations, spreadsheets, and structured data in corporate databases, while web pages are undoubtedly a part of that distributed knowledge store. Desktop search is the individual version of corporate search, where the information sources are the documents saved on a user's PC, including emails and recently visited websites. Without centralized control, peer-to-peer search involves locating data in networks of nodes or computers. This kind of search initially served as a music file-sharing platform, but it can now be used to any group of people who have similar interests or, in the case of mobile devices, a local area.Search and related information retrieval techniques are utilized in a variety of industries, including advertising, intelligence gathering, science, healthcare, customer service, real estate, and others. Any application that uses a collection of unstructured data, such as text or other types of information, will need to organize and search that data.

There are other text-based tasks that are researched in information retrieval besides search based on a user query (sometimes referred to as ad hoc search because the range of possible queries is broad and not prespecified). Filtering, categorization, and question-answering are additional duties. Based on a person's interests, filtering or monitoring involves finding stories that are relevant to them and sending them an alert by email or another method. A predetermined set of labels or classifications are used in classification or categorization, which automatically applies those labels to documents. Similar to search, question answering focuses on more specific queries such "What is the height of Mt. Everest?". Instead of returning a list of documents, the aim of question answering is to return a specific response that was found in the text. Some of these components or dimensions of the information retrieval field are summarized in Table 1.

Table 1: Some dimensions of information retrieval

Examples of Content	Examples of Applications	Examples of Tasks
Text	Web search	Ad hoc search
Images	Vertical search	Filtering
Video	Enterprise search	Classification
Scanned documents	Desktop search	Question answering
Audio	Peer-to-peer search	
Music		

1.2 COMPONENTS OF IR

The main parts of an IR system are shown in Figure 1. A user's information demand, which underpins and motivates the search process, exists prior to performing a search. When this information need is offered

in writing form as a test collection for IR evaluation, we occasionally refer to it as a topic. The user creates and submits a query to the IR system as a result of her information need. This search usually only has one or two terms, with two to three terms being normal for a Web search. Because a query term might not actually be a word at all, we use "term" instead of "word" in this sentence.A query term could be a date, a number, a musical note, or a phrase, depending on the information required. Query phrases may also be allowed to contain wildcard operators and other partial-match operators. The term "inform," for instance, might refer to any word beginning with that prefix (e.g., "informs," "informs," "informal," "informant," "informative," etc.).

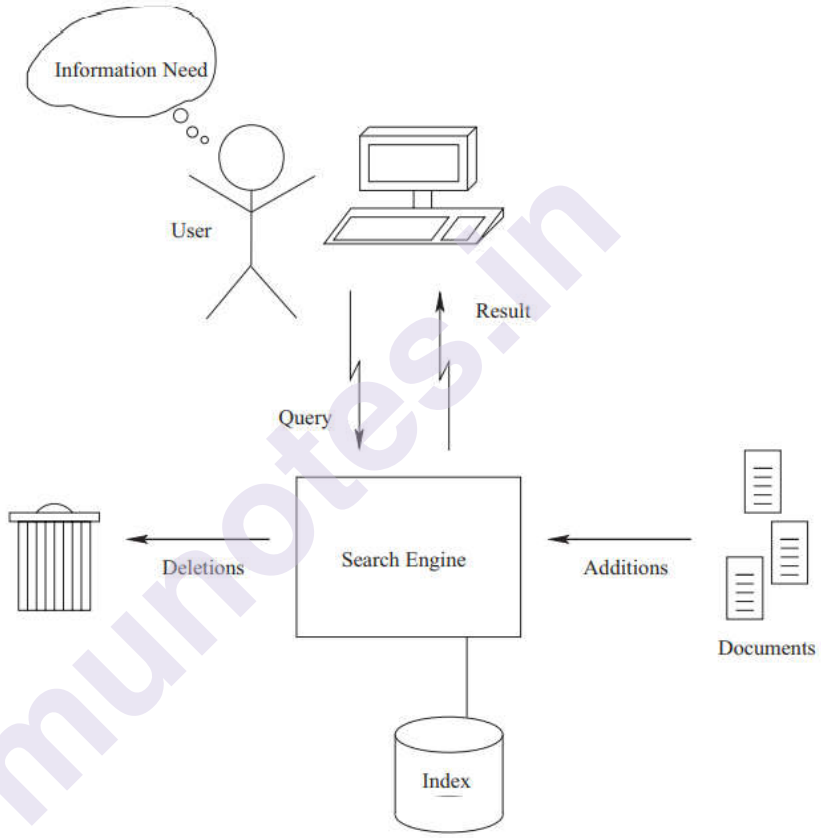

Figure 1: Components of an IR system

IR systems generally allow a richer query syntax, frequently with complex Boolean and pattern matching operators, even though users typically make simple keyword searches. These tools can be used to confine a search to a certain Web site, to establish restrictions on fields like author and title, or to use additional filters, limiting the search to a portion of the collection. When these more complex query facilities are necessary, a user interface mediator between the user and the IR system, streamlining the query-creation process.

A search engine, which may be running on the user's local workstation, on a sizable cluster of machines in a far-off location, or anywhere in between, processes the user's query. The management and manipulation of an inverted index for a document collection is a key responsibility of a search

engine. The main data structure that the search engine uses to rank relevance is this index. An inverted index's primary purpose is to map out the relationships between terms and the locations in the collection where they appear. It is important to take care that index access and update operations are carried out effectively because the size of an inverted list is on the same scale as the document collection itself.

The search engine keeps index collecting statistics, such as the number of documents containing each term and the length of each document, to help relevance ranking algorithms. In order to provide the user with relevant results, the search engine typically has access to the source content of the documents.

The search engine accepts queries from its users, analyses these questions, and then produces prioritized lists of results using the inverted index, collection statistics, and other data. The search engine calculates a score, sometimes referred to as a retrieval status value (RSV), for each document to perform relevancy ranking. Following the score-based sorting of the documents, additional processing, such as the elimination of redundant or duplicate results, may be applied to the result list. One or two results from a single host or domain, for instance, might be reported by a web search engine, with the remaining pages being replaced by pages from other sources. One of the most essential issues in the industry is the scoring of documents in relation to a user's query.

1.3 ISSUES RELATED TO IR

Since the 1960s, when experiments were conducted on document collections totaling around 1.5 gigabytes of text, information retrieval experts have concentrated on a few core challenges that are still vitally essential in the era of commercial web search engines dealing with billions of web pages. Following are few of the issues related to IR:

1] Relevance: A key idea in information retrieval is relevance. A relevant document, broadly speaking, contains the details that a user was seeking when she entered a search term into the search engine. Although it may seem straightforward, determining whether a given document is relevant depends on a variety of factors. When formulating algorithms for comparing text and rating texts, these elements must be taken into consideration.In a database system or when using the grep utility in Unix, comparing the text of a query with the text of a document and seeking for an exact match only yields very unreliable results in terms of relevancy. The fact that language may be used to describe the same ideas in numerous ways, frequently using very different words, is one clear explanation for this. The vocabulary mismatch problem in information retrieval is what this is known as.

Separating topical relevance from user relevance is also crucial. If a text document is on the same topic as a query, it is topically relevant. A news report on a tornado in Kansas, for instance, would be topically pertinent to the search term "severe weather incidents." However, if the questioner

(commonly referred to as the user) has already seen the story, it is five years old, it is in Chinese from a Chinese news source, or it is in another language, she may not find it to be relevant. These extra elements of the story are taken into account when determining user relevance.

Researchers suggest retrieval models and evaluate their effectiveness in order to overcome the relevance issue. A retrieval model is a formal illustration of how a query and a document are matched. It serves as the cornerstone of the ranking algorithm that a search engine employs to generate the ranked list of pages. Finding documents that the user who submitted the query is likely to find relevant is the goal of a decent retrieval model. Some retrieval models place a greater emphasis on topical relevance, but a search engine used in a live setting needs to employ ranking algorithms that take user relevance into account.

The fact that information retrieval models often model the statistical rather than the linguistic structure of text is an intriguing aspect of these models. This implies, for instance, that word frequency countsrather than whether a word is a noun or an adjectiveare often much more important to ranking algorithms. Linguistic features are included in more sophisticated models, but they often play a supporting role. H.P. Luhn, another information retrieval pioneer, introduced the use of word frequency data to represent text in the 1950s. It took until the 1990s for this perspective on text to catch on in other branches of computer science, like natural language processing.

2] Evaluation: Since a document ranking's quality depends on how well it meets a user's expectations, early evaluation metrics and experimental techniques for gathering this data and applying it to evaluate ranking algorithms were required. Early in the 1960s, Cyril Cleverdon took the lead in creating evaluation techniques, and today, precision and recall are still widely utilized. The percentage of papers that are relevantly returned is known as precision, and it is a relatively simple unit of measurement. The percentage of pertinent papers that are retrieved is known as recall. The recall measure operates under the presumption that all pertinent documents for a particular query are known.In a web search context, such an assumption is obviously problematic, but this strategy can be helpful for smaller test collections of documents. A sample of common queries, a list of pertinent documents for each query, and a collection of text documents make up an information retrieval experiment test collection (the relevance judgments). The test collections connected to the TREC assessment forum are the most well-known.

The evaluation of retrieval models and search engines is a very active field right now, and a lot of the attention is being paid to the usage of massive amounts of log data from user interactions, like clickthrough data, which records the documents that were clicked during a search session. Search engine businesses still rely on relevance judgements in addition to log data to guarantee the accuracy of their results, despite the fact that clickthrough and other log data have a strong correlation with relevance and can be used to evaluate search.

3] Emphasis on users and their information needs: Given that the evaluation of search is user-centered, this ought to be obvious. In other words, search engine users are the final arbiters of value. Numerous research has been conducted as a result on how users interact with search engines, and in particular, approaches have been created to assist users in expressing their information demands. The primary motivation behind a person's search engine query is an information requirement. Text queries are frequently inadequate explanations of what the user actually needs, as opposed to a request to a database system, such as for the balance of a bank account.If you search for "cats," you can be looking for information on where to buy cats or a synopsis of the Broadway show. One-word queries are extremely prevalent in web search, despite their lack of specificity. The initial query is refined using interaction and context to provide higher ranked lists using techniques like query expansion, query suggestion, and relevance feedback.

1.4 BOOLEAN RETRIEVAL

Explicit support for Boolean searches is crucial in particular application domains like digital libraries and the legal sector, in addition to the implicit Boolean filters used by Web search engines. Boolean retrieval delivers sets of documents rather than ranked lists, in contrast to ranked retrieval. A term t is regarded by the Boolean retrieval model as specifying the collection of documents that contain it. Boolean queries, which are regarded as operations on these sets, are built using the common Boolean operators AND, OR, and NOT as follows:

A AND B: intersection of A and B ($A \cap B$)

A OR B: union of A and B ($A \cup B$)

NOT A: complement of A with respect to the document collection ($\bar{A}$)

where A and B are terms or other Boolean queries.

Consider table 2 as an example.

Table 2:Text fragment from Shakespeare's Romeo and Juliet, act I, scene 1

Document ID	Document Content
1	Do you quarrel, sir?
2	Quarrel sir! no, sir!
3	If you do, sir, I am for you: I serve as good a man as you.
4	No better.
5	Well, sir.

For example, over the collection in Table 2, the query

("quarrel" OR "sir") AND "you"

specifies the set {1, 3}, whereas

the query ("quarrel" OR "sir") AND NOT "you"

specifies the set {2, 5}.

The phrase searching algorithm of Figure 2 and the cover finding method of Figure 3 are two variations of our approach for resolving Boolean inquiries. The technique finds candidate answers to a Boolean query, where each candidate answer is a set of documents that individually meet the Boolean question but do not include any smaller sets of documents that also fulfil the query. This single document satisfies the query and belongs in the result set when the range represented by a candidate solution has a length of 1.

```
    nextPhrase (t1t2...tn, position) ≡
1       v ← position
2       for i ← 1 to n do
3           v ← next(ti, v)
4       if v = ∞ then
5           return [∞, ∞]
6       u ← v
7       for i ← n − 1 down to 1 do
8           u ← prev(ti, u)
9       if v − u = n − 1 then
10          return [u, v]
11      else
12          return nextPhrase(t1t2...tn, u)
```

Figure 2: Function to locate the first occurrence of a phrase after a given position. The function calls the **next** and **prev** methods of the inverted index ADT and returns an interval in the text collection as a result

```
    nextCover (⟨t1, ..., tn⟩, position) ≡
1       v ← max1≤i≤n(next(ti, position))
2       if v = ∞ then
3           return [∞, ∞]
4       u ← min1≤i≤n(prev(ti, v + 1))
5       if docid(u) = docid(v) then
6           return [u, v]
7       else
8           return nextCover(⟨t1, ..., tn⟩, u)
```

Figure 3: Function to locate the next occurrence of a cover for the term vector <t1, ..., tn> after a given position

Both of the aforementioned algorithms have the same fundamental mode of operation. Lines 1-6 of the phrase search algorithm identify a range that contains every term in the phrase in the order that it appears, such that no smaller range included within it also contains every term in the phrase. Lines 1-4 similarly identify all the terms as closely as possible in the cover searching method. Then, an additional constraint is imposed to both methods.

To simplify our definition of our Boolean search algorithm, we define two functions that operate over Boolean queries, extending the **nextDoc** and **prevDoc** methods of schema-dependent inverted indices.

docRight(Q, u) — end point of the first candidate solution to Q starting after document u

docLeft(Q, v) — start point of the last candidate solution to Q ending before document v

For terms we define:

$$\begin{aligned} \textbf{docRight}(t, u) &\equiv \textbf{nextDoc}(t, u) \\ \textbf{docLeft}(t, v) &\equiv \textbf{prevDoc}(t, v) \end{aligned}$$

and for the AND and OR operators we define:

$$\begin{aligned} \textbf{docRight}(A \textbf{ AND } B, u) &\equiv \max(\textbf{docRight}(A, u), \textbf{docRight}(B, u)) \\ \textbf{docLeft}(A \textbf{ AND } B, v) &\equiv \min(\textbf{docLeft}(A, v), \textbf{docLeft}(B, v)) \\ \textbf{docRight}(A \textbf{ OR } B, u) &\equiv \min(\textbf{docRight}(A, u), \textbf{docRight}(B, u)) \\ \textbf{docLeft}(A \textbf{ OR } B, v) &\equiv \max(\textbf{docLeft}(A, v), \textbf{docLeft}(B, v)) \end{aligned}$$

To determine the result for a given query, these definitions are applied recursively. For example:

$$\begin{aligned} &\textbf{docRight}((\text{“quarrel” OR “sir”}) \text{ AND “you”}, 1) \\ &\quad \equiv \max(\textbf{docRight}(\text{“quarrel” OR “sir”}, 1), \textbf{docRight}(\text{“you”}, 1)) \\ &\quad \equiv \max(\min(\textbf{docRight}(\text{“quarrel”}, 1), \textbf{docRight}(\text{“sir”}, 1)), \textbf{nextDoc}(\text{“you”}, 1)) \\ &\quad \equiv \max(\min(\textbf{nextDoc}(\text{“quarrel”}, 1), \textbf{nextDoc}(\text{“sir”}, 1)), 3) \\ &\quad \equiv \max(\min(2, 2), 3) \\ &\quad \equiv 3 \end{aligned}$$

$$\begin{aligned} &\textbf{docLeft}((\text{“quarrel” OR “sir”}) \text{ AND “you”}, 4) \\ &\quad \equiv \min(\textbf{docLeft}(\text{“quarrel” OR “sir”}, 4), \textbf{docLeft}(\text{“you”}, 4)) \\ &\quad \equiv \min(\max(\textbf{docLeft}(\text{“quarrel”}, 4), \textbf{docLeft}(\text{“sir”}, 4)), \textbf{prevDoc}(\text{“you”}, 4)) \\ &\quad \equiv \min(\max(\textbf{prevDoc}(\text{“quarrel”}, 4), \textbf{prevDoc}(\text{“sir”}, 4)), 3) \\ &\quad \equiv \min(\max(2, 3), 3) \\ &\quad \equiv 3 \end{aligned}$$

Definitions for the NOT operator are more difficult, so we wait to discuss it until after the core algorithm has been introduced.

The **nextSolution** function, shown in Figure 4, finds the following Boolean question solution after a specified point.

```
nextSolution (Q, position) ≡
1    v ← docRight(Q, position)
2    if v = ∞ then
3        return ∞
4    u ← docLeft(Q, v + 1)
5    if u = v then
6        return u
7    else
8        return nextSolution (Q, v)
```

Figure 4: Function to locate the next solution to the Boolean query Q after a given position. The function **nextSolution** calls **docRight** and **docLeft** to generate a candidate solution. These functions make recursive calls that depend on the structure of the query

For the purpose of producing a potential solution, the function calls **docRight** and **docLeft**. This potential answer can be found in the interval [u,v] right after line 4. The potential solution is returned if it only consists of one document. Otherwise, a recursive call is made by the function. All answers to the Boolean inquiry Q may be produced by the following given this function:

```
u ← −∞
while u < ∞ do
    u ← nextSolution(Q, u)
    if u < ∞ then
        report docid(u)
```

The temporal complexity of this algorithm, using a galloping search implementation of nextDoc and prevDoc, is O(n·l·log(L/l)), where n is the number of terms in the query. l and L reflect the lengths of the shortest and longest posting lists of the terms in the query as measured by the number of documents if a docid or frequency index is utilised and positional information is not kept in the index. The justification needed to prove this time complexity is comparable to that of our proximity ranking algorithm and word search algorithm.The temporal complexity changes to O(n·κ·log(L/κ)) when expressed in terms of the quantity of candidate solutions, which illustrates the adaptive character of the algorithm. Although the call to the **docLeft** function in line 4 of the algorithm can be eliminated, by clearly defining a potential answer, it aids in our analysis of the algorithm's complexity.

We didn't take into account the NOT operator while defining **docRight** and **docLeft**. In fact, implementing the generalized versions of these functions is not required in order to use the NOT operator. Instead, a query can be transformed by applying De Morgan's laws, which will push any NOT operators inside until they are very next to the query terms:

$$\textbf{NOT}\ (A\ \textbf{AND}\ B) \equiv \textbf{NOT}\ A\ \textbf{OR}\ \textbf{NOT}\ B$$
$$\textbf{NOT}\ (A\ \textbf{OR}\ B) \equiv \textbf{NOT}\ A\ \textbf{AND}\ \textbf{NOT}\ B$$

For example, the query

"william" AND "shakespeare" AND NOT ("marlowe" OR "bacon")

would be transformed into

"william" AND "shakespeare" AND (NOT "marlowe" AND NOT "bacon").

This transformation does not alter the quantity of AND and OR operators, and as a result, does not alter the quantity of terms in the query (n). We are left with a query that contains expressions of the form **NOT** t, where t is a word, after properly applying De Morgan's principles. We need comparable definitions of **docRight** and **docLeft** in order to process queries with expressions of this type. These definitions could be expressed in terms of **nextDoc** and **prevDoc**.

```
docRight(NOT t, u) ≡
    u' ← nextDoc(t, u)
    while u' = u + 1 do
        u ← u'
        u' ← nextDoc(t, u)
    return u + 1
```

Unfortunately, this strategy raises the possibility of inefficiencies. Although this definition will work well when few documents contain the term "t," it may perform poorly when most documents contain the term "t," effectively returning to the linear scan of the postings list that galloping search avoided. Additionally, the comparable implementation of docLeft(NOT t, v) necessitates a backward scan of the postings list, which is against the rule required to benefit from galloping search.

Instead, we may implement the NOT operator directly over the data structures, extending the methods supported by our inverted index with explicit methods for **nextDoc(NOT t, u)** and **prevDoc(NOT t, v)**.

1.5 DICTIONARIES AND TOLERANT RETRIEVAL

1.5.1 Search structures for dictionary

Our initial objective is to check whether each query term is present in the lexicon and, if it is, find the reference to the associated postings given an inverted index and a query. There are two main groups of solutions for this vocabulary lookup action: hashing and search trees. The dictionary is a traditional data structure that is used in this operation. The entries in the lexicon (in our case, terms) are typically referred to as keys in the literature on data structures.A number of questions determine the best solution (hashing or search trees): (1) How many keys are likely to be in our possession? (2) Is it likely that the number will stay the same or change significantly, and if it does, are we likely to see only new keys added or also some keys removed from the dictionary? (3) How frequently will different keys be accessed, on average?

In some search engines, hashing has been employed for dictionary searches. Each word in the dictionary is hashed into an integer over a sufficiently enough area to make hash collisions improbable; if any, they are resolved using auxiliary structures that can be labor-intensive to maintain. 1 We hash each query phrase individually at query time, following a pointer to the associated postings, and accounting for any logic for handling hash clashes.Minor variations of a search term (such as the accented and unaccented forms of a word like "resume") might have extremely different hash values, so it is difficult to discover them. Finally, a hash function created to meet present demands may not be enough in a few years in a context (like the Web) where the vocabulary is expanding.

Many of these problems are resolved by search trees, which, for example, allow us to enumerate all lexical phrases beginning with automat. The binary tree, which contains two children at each internal node, is the most well-known search tree.At the tree's base, a phrase is sought after. Each internal node (including the root) represents a binary test, the results of which determine which of the two subtrees lies below that node in the internal tree. An illustration of a binary search tree used for a dictionary may be found in Figure 5. The balance of the tree is crucial for efficient search (with a number of comparisons that is O(log M)). To mitigate rebalancing, one approach is to allow the number of sub-trees under an internal node to vary in a fixed interval.

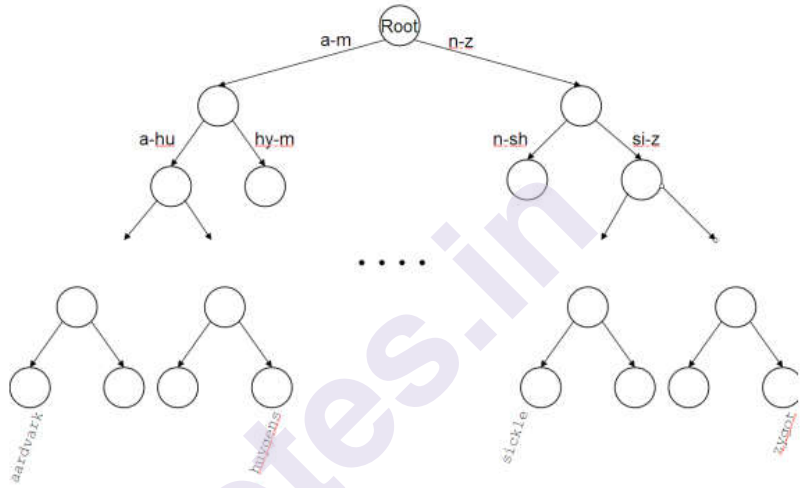

Figure 5:A binary search tree. In this example the branch at the root partitions vocabulary terms into two subtrees, those whose first letter is between a and m, and the rest

1.5.2 Wildcard queries

Wildcard queries are used in any of the following situations: (1) the useris uncertain of the spelling of a query term (e.g., Sydney vs. Sidney, whichleads to the wildcard query S*dney); (2) the user is aware of multiple variants of spelling a term and (consciously) seeks documents containing any ofthe variants (e.g., color vs. colour); (3) the user seeks documents containingvariants of a term that would be caught by stemming, but is unsure whetherthe search engine performs stemming (e.g., judicial vs. judiciary, leading to thewildcard query judicia*); (4) the user is uncertain of the correct rendition of aforeign word or phrase (e.g., the query Universit* Stuttgart).

A trailing wildcard query, such as mon*, is one in which the wildcard * sign appears just once, at the conclusion of the search term. It is simple to handle trailing wildcard searches by using a search tree on the dictionary. By moving down the tree while following the symbols m, o, and n in turn, we may eventually enumerate the set W of entries in the dictionary that have the prefix mon.Finally, we obtain all documents containing any term in W using |W| lookups on the common inverted index.

But what about searches that use wildcards, in which the * sign is not required to appear at the end of the search string? We briefly generalize trailing wildcard queries before dealing with this general instance. Consider leading wildcard queries or *mon-style queries first. Think of a reverse B-tree on the dictionary, where each root-to-leaf path represents a word in the dictionary spelled backwards. For example, the term "lemon" would be represented by the path root-n-o-m-e-l in the B-tree. The next step is to go down the reverse B-tree to find all terms R in the lexicon that begin with a specific prefix.

In reality, we can handle a case that is even more generic by combining a conventional B-tree with a reverse B-tree: wildcard queries with a single * sign, like se*mon. To do this, we first enumerate the set W of dictionary terms beginning with the prefix se using the ordinary B-tree, and then we enumerate the set R of terms ending in the suffix mon using the reverse B-tree. The collection of phrases that start with the prefix se and end with the suffix mon is then obtained by taking the intersection $W \cap R$ of these two sets. Finally, we obtain all documents that contain any of the terms in this intersection using the conventional inverted index.We can thus handle wildcard queries that contain a single * symbol using two B-trees, the normal B-tree and a reverse B-tree.

1.6 SUMMARY

Calvin Mooers originated the phrase "information retrieval" in 1950. From 1961 onwards, when computers were developed for information handling, it gained appeal among researchers. The subsequent definition of information retrieval included the extraction of bibliographic data from databases of archived documents. But those document retrieval systems were actually information retrieval systems. These were created to discover the existence (or absence) of bibliographic documents pertinent to a user's search.To put it another way, early IRS were built to return a complete documenta book, an article, etc.in answer to a search query. Although the IRS still operates in this manner today, numerous cutting-edge design methods have been created and implemented over time. The meaning of information retrieval has evolved over time, and scholars and information specialists have used many terminologies to describe it. Information access, text retrieval, information representation and retrieval, information processing and retrieval, and information storage and retrieval are only a few of them. We have also discussed about the components used in the information retrieval system along with the issues associated with it.

1.7 LIST OF REFERENCES

1] Introduction to Information Retrieval, C. Manning, P. Raghavan, and H. Schutze, Cambridge University Press, 2008

2] Modern Information Retrieval: The Concepts and Technology behind Search, Ricardo Baeza -Yates and Berthier Ribeiro – Neto, 2nd Edition, ACM Press Books 2011.

3] Search Engines: Information Retrieval in Practice, Bruce Croft, Donald Metzler and TrevorStrohman, 1st Edition, Pearson, 2009.

4] Information Retrieval Implementing and Evaluating Search Engines, Stefan Buttcher, Charles L. A. Clarke and Gordon V. Cormack, The MIT Press; Reprint edition (February 12, 2016).

1.8 UNIT END EXERCISES

1] Define information retrieval system along with its component.

2] Discuss the history of information retrieval system.

3] What are the different issues of information retrieval system.

4] Explain the Boolean retrieval.

5] Discuss on dictionaries and tolerant retrieval.

6] Explain the search structures for dictionary.

7] What are the wildcard queries.

2

LINK ANALYSIS AND SPECIALIZED SEARCH

Unit Structure

2.0 OBJECTIVES

After completion of this module, you will learn:

- Definition of link analysis
- To do link analysis, use graphs.
- Show how PageRank is calculated, discuss several
- PageRank calculation methods, and use MapReduce to do it.

2.1 INTRODUCTION

In this part of the book, we are going to delve into the intriguing area of link analysis, which is a strong tool for studying the links between entities in a network. In particular, we are going to look at how to use link analysis. The purpose of link analysis is to identify patterns, trends, and other significant insights by investigating the connections that exist between the nodes that make up a network. The PageRank algorithm, which determines the relative relevance of web sites based on the number and quality of incoming and outgoing links to those pages, is one of the link analysis methods that is utilised by the largest number of people.

The PageRank algorithm computes a rank for a website based on the chance that the user would go to various links by using graphs to represent the web. The huge size of the web necessitates the use of procedures using matrices of very large sizes in order to complete the calculation. As a result, the MapReduce programming paradigm is applied on top of the distributed file system via the PageRank algorithm.

In this section, we will go over the fundamentals of link analysis, demonstrate how graphs may be used to do link analysis, and discuss how PageRank is calculated. In addition to this, we will learn about the various methods that may be used to calculate PageRank, as well as how MapReduce can be used to carry out this calculation. This lesson will give a good basis for understanding this significant area, whether you are a student, researcher, or just inquisitive about the realm of link analysis.

2.2 LINK ANALYSIS

The use of hyperlinks for ranking online search results is the main topic of this chapter. Web search engines consider a variety of elements when calculating a web page's overall score for a specific query, one of which is link analysis. Page rank and HITS are two different techniques for link analysis.

Network theory uses the data analysis method of link analysis to examine the web graph's connections. A collection of nodes and a set of edges, links, or intersections between nodes make up a graph. The graphs are used everywhere, including on social networking platforms like Facebook, Twitter, and others.

2.2.1 Why Link Analysis Is Done?

The goal of link analysis is to establish connections between data points so that the data may be seen as networks or informational networks. For instance, on the Internet, computers or routers may interact with one another via a dynamic network of nodes that stand in for the computers and routers. Physical connections between these devices make up the network's edges. The web is another illustration, which may be seen as a graph.

2.2.2 The Web as a graph

A web made out of static HTML pages linked by links in the form of a directed graph, where each page is a node and each link are a directed edge.

Figure 1.2.1 A link connects two web graph nodes. [4]

Figure 2.2.1 shows two web graph nodes A and B, each of which represents a web page, and which are linked by a link from A to B. The collection of all such nodes and directed edges is referred to as the web graph. The Figure shows that there is text around the hyperlink's source on page A. Anchor text is the text that is typically included in the href attribute of the <a> (for anchor) element that encodes the hyperlink in the HTML code of page A.

2.2.3 Graph-based representation of the World Wide Web (WWW)

A directed graph may be used to depict the Web. As a result, web pages will be the nodes in the graph. As a result, each website will represent a node in this network, and direct linkages between them will act as hyperlinks. The associations between the hyperlinks may be utilised to build a network.

2.2.4 Types of links

1. Inbound links:

Links leading to a website from outside sources are called inbound links.

One technique to raise a site's overall Page Rank is via inlinks. Inlinks do not penalise websites.

2. Outbound links:

A page may connect to other pages on the same website or to other websites via outbound links.

3. Dangling links:

Links that point to any page but have no outgoing connections are known as dangling links.

2.2.5 Link spam

Unethical web page designers may attempt to generate pointless connections in order to elevate one of their web pages in the search engine results since it is commonly known that commercial search engines utilise methods like Page Rank and anchor text extraction. It's known as link spam.

2.3 HUBSAND AUTHORITIES

Every online page is given two scores, not one, as is the case with the PageRank algorithm, which is the primary tenet of the hubs and authority models. The hub score is one type score, while the authority score is the other type. The total of the scaled hub values pointing to a certain web page constitutes an authority value. The scaled authority values of the pages that a hub points to make up its worth.

The well-known, highly recommended responses to the searches will be referred to as the authorities for the question. These were the kind of sites we were initially looking for. The hubs for the query will be referred to as the high-value lists. Now, we attempt to assess the worth of each page p as a prospective authority and hub by giving it two numerical scores: *auth*(*p*) and *hub*(*p*). Each of these begins with a value of 1, demonstrating that we are first unsure of which falls under which of these categories. Now, voting is as easy as the following. We utilise the quality of the hubs to improve our estimations of the authority's quality.

Authority Update Rule: Update *auth*(*p*) to be the total of all hub scores for pages that point to page *p*.

On the other hand, the list-finding method, in which we improve our estimations of the quality of the hubs by the caliber of the authorities, is as follows:

Hub Update Rule: Update *hub*(*p*) to be the total of all the authority ratings of the pages it points to for each page *p*.

Keep in mind that a single application of the *Authority Updating Rule* is equivalent to the first casting of votes by in-links (beginning from a setup where all scores are originally 1). The outcomes of the original list-finding method are obtained by applying the *Authority Update Rule* once, then the *Hub Update Rule* once. In general, the Principle of Repeated Improvement states that we should just use these criteria alternately to get better estimates.

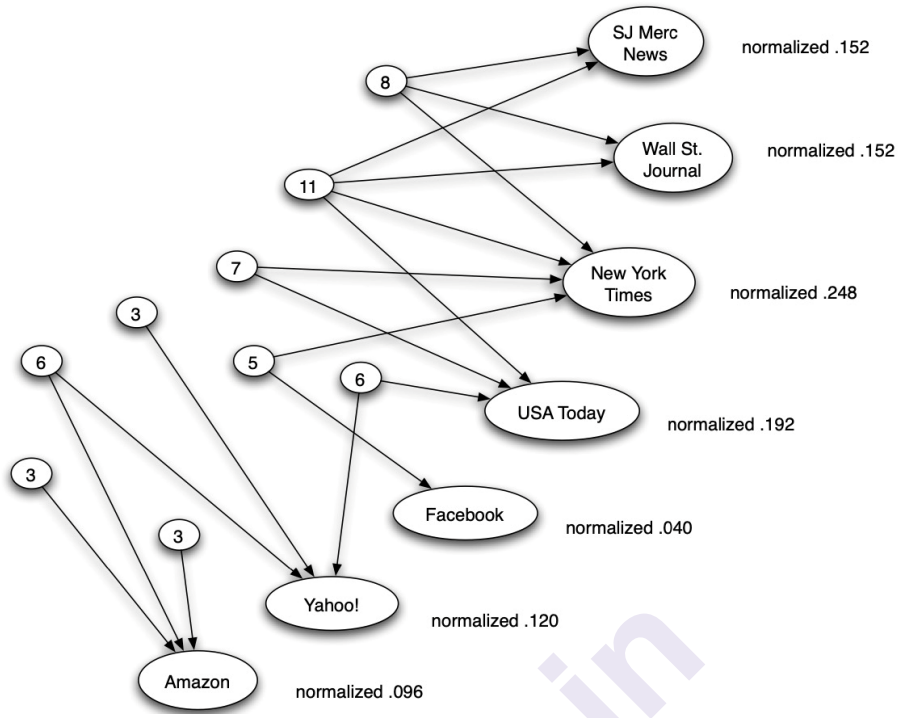

Figure 2.3.1 Re-weighting votes after normalizing for the query "newspapers."[1]

• All hub scores and authority scores are set to 1 at the beginning.

• We choose a set of steps, *k*.

• After that, we execute a series of *k* hub-authority upgrades. Every update function as follows:

Apply the Hub Update Rule to the final set of scores after first applying the *Authority Update Rule* to the current set of results.

• The hub and authority scores may ultimately entail extremely big numbers. But, because we are only interested in their relative sizes, we may normalise them by dividing each authority score by the total of all authority scores and each hub score by the total of all hub scores.

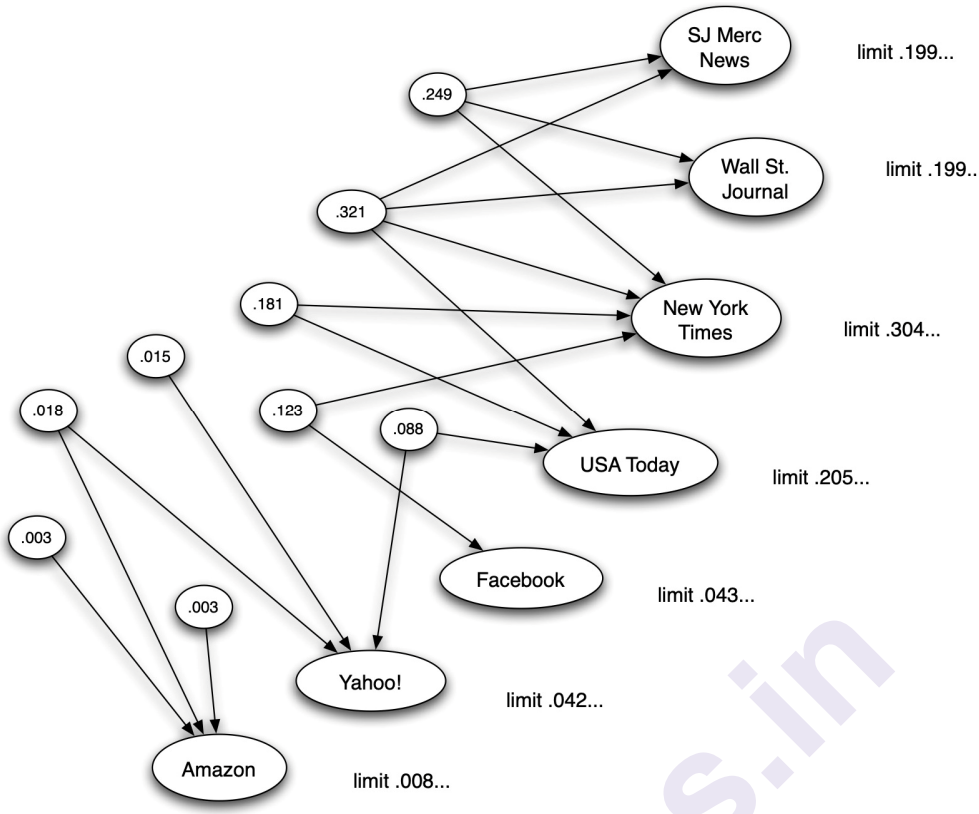

Figure 2.3.2 Limiting hub and authority values for the query "newspapers."[1]

The most important details in this text are that the normalized values of *k* actually converge to limits as *k* goes to infinity, and that the limiting hub and authority values are a property of the link structure, not of the initial estimates we use to start the process of computing them. This means that the limiting hub and authority values remain unchanged if we apply the *Authority Update Rule* or the *Hub Update Rule,* reflecting the balance between hubs and authorities that provided the initial intuition for them. This means that the authority score is proportional to the hub scores of the pages that point to you, and the hub score is proportional to the authority scores of the pages you point to.

2.4 PAGE RANK AND HITS ALGORITHMS

The PageRank algorithm was created for the Google search engine's first rollout. The PageRank is a method for determining the significance of online sites in a large web graph.

PageRank – Basic Concept: As many links as a page or node has, that's how important it is. Not every in-link is the same type of link. Links that come from important pages are more valuable. How important a page isbased on how important other pages are that link to it. So, the importance of a given page with a point depends on the importance of the other pages with points, and that page's importance is then passed on through the graph.

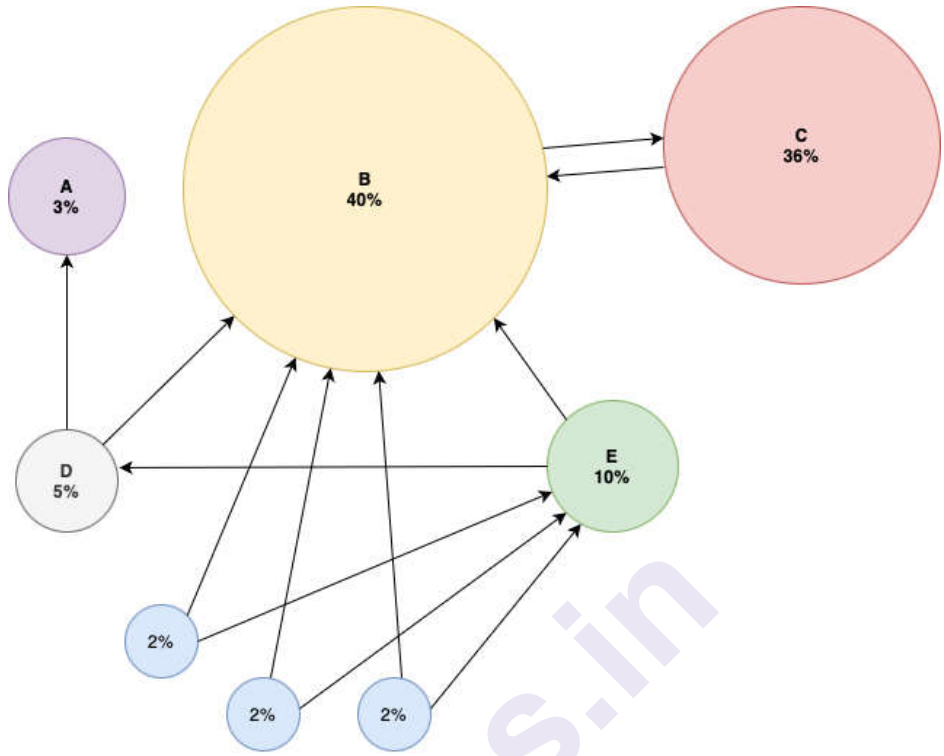

Figure 2.4.1 PageRank Scores [1]

Figure 2.4.1 shows a graph with a set of directed edges, where the size of a node is related to its PageRank score. The PageRank scores are set up so that when added up, they equal 100.

Here is a summary of the page rank scores:

The node B has a very high PageRank score because a lot of another page's link to it. The node C also has a fairly high PageRank score, even though it only gets one incoming link. This is because the very important node B points in that direction.

- The very small purple link nodes have a low PageRank score because they are pointed to by a website with a low PageRank.
- The PageRank score of node D is higher than that of node A because D points to A.
- The PageRank score of node E is kind of in the middle, and E points to node B, and so on.

Figure 2.4.1 shows that the PageRank are pretty easy to understand, and they match how we think a node in our graph is important.

In this section, we talk about the simple recursive formulation and the flow model, which are two important ways to figure out PageRank.

The Simple Recursive Formulation method will be used to find PageRank Scores, in which:

- Each link is treated as a vote, and the importance of a given vote is proportional to the importance of the source web page that casts this vote or creates a link to the destination.
- Let's say that page j has an importance of rj.
- There are n links that leave page j.
- This page j's importance rj is pretty much split evenly among all the links that lead away from it.
- Each link gets rj divided by n votes i.e., rj/n votes.

This is how j and the pages it points to share the importance. And in the same way, you can figure out how important a node is by adding up the votes on its in-links.

Figure 2 is a simple graph that shows how many votes a page called "j" got. Using the following formula, you can figure out the score of node j as the sum of the votes it got from ri and rk:

$$rj=ri/2+rk/3$$

This is because there are 3 out links on page I and 4 out links on page k. Along with the three-outgoing links, the score of page j is also shared outside of page j. So, each of these links gets one-third of the importance of node j.

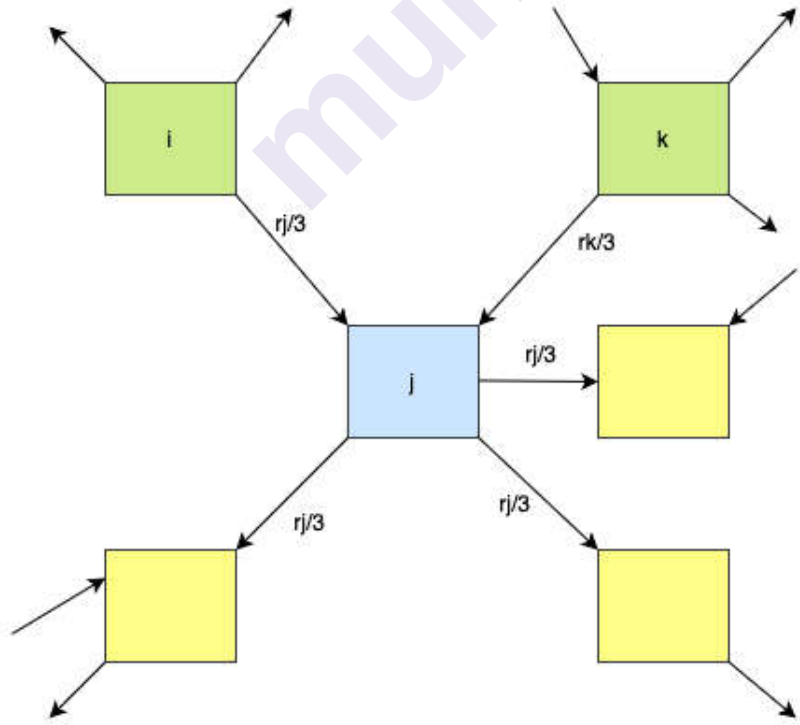

Figure 2.4.2 A Graph to calculate score of pages [1]

This is basically how each node finds out how important the pages that link to it are and spreads that information to the nodes around it.

The Flow Model: This model shows how votes move through a network. So, this is why it's called a flow model or the flow formulation of PageRank. To help you understand the idea, let's look at a web graph of the WWW with only three web pages called a, b, and c. (Please refer to Figure 2.4.3).

• A has a self-link that goes to b.

• B links to both A and C.

• C only links back to b.

When an important page votes, it counts more than when a normal page vote. So, a page is more important if another important page's link to it.

Formula to compute PageRank:

$$r_j = \sum_{i \to j} r_i d_i$$

Where di is the node i's out-degree. You may remember that a node's degree is the number of links that leave it. So, page j's importance score is just the sum of all the pages i that point to it. To figure out how important a page i is, you divide its importance by its out degree.

This means that we get a different equation for each node in the network based on the number of links. For example, the importance of node an in the network is just the importance of a divided by 2 plus the importance of b divided by 2. Because node a has two links that go out, and node b has the same number. This means that the three equations for Figure 2.4.3, one for each node, would be:

$r_a = r_a/2 + r_b/2$

$r_b = r_a/2 + r_c$

$r_c = r_b/2$

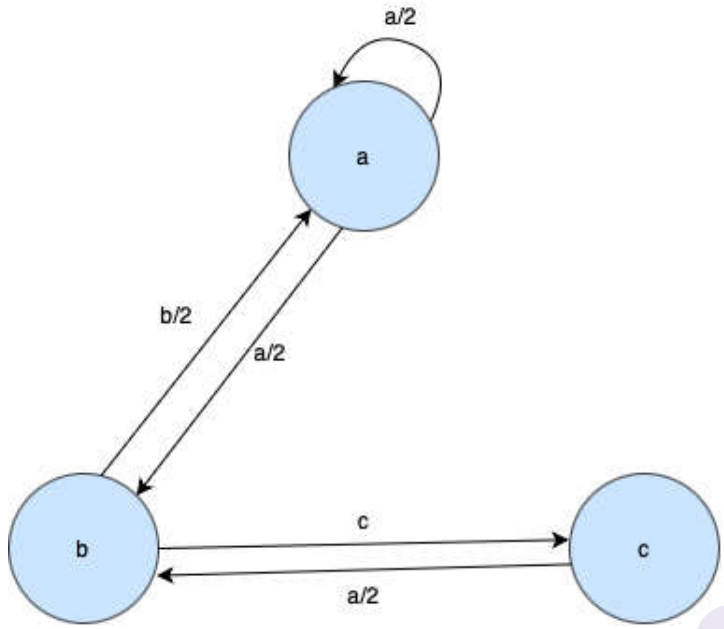

Figure 2.4.3A very simple web graph [1]

The problem is that none of these three equations has a single answer. So, we need to add another constraint if we want the solution to be unique. Let's add a rule: the sum of all the PageRank scores should be 2. This is shown by the equation below:

$r_a + r_b + r_c = 1$

You can solve these equations and find the solution to the PageRank scores, which is:

$r_a = 2/5;\ r_b = 2/5;\ r_c = 1/5$

Here is pseudocode for the **PageRank Algorithm**:

Input:

- A directed graph G = (V, E),

where V is the set of nodes (web pages) and E is the set of edges

(links between pages)

- A damping factor d (usually set to 0.85)

- A threshold value ε

(usually set to a small value, such as 0.0001)

Output:- A vector PR containing the PageRank scores

for each node in V

2. Initialize all PageRank scores to 1/|V|

PR(i) = 1/|V|, where i∈ V

2. Repeat until convergence:

For each node i∈ V, do the following:

a. Compute the sum of incoming PageRank scores to node i:

In(i) = Σ{j ∈ In(i)} PR(j)/OutDegree(j)

b. Update the PageRank score for node i:

PR(i) = (1-d)/|V| + d*In(i)

Check for convergence:

If ||PR - previous_PR|| < ε, exit the loop

Else, set previous_PR = PR and repeat from step 2

3. Return the PageRank scores vector PR

In this pseudocode, the PageRank score for a node i is calculated based on the incoming links (In(i)) and the PageRank scores of the nodes that link to it. The PageRank scores are then updated based on a damping factor (d) that determines the probability of a random surfer clicking on a link, and a threshold value (ε) that determines when the algorithm has converged. The algorithm continues to iterate until the PageRank scores converge, at which point the final scores are returned as a vector PR

The HITS algorithm

The HITS algorithm, or Hypertext Induced Topic Selection, is the name of the hubs and authority-based method that underlies this one.

- Similar to how we computed PageRank, it is used to assess the significance of pages or documents.
- For instance, we need to locate a collection of excellent newspaper websites.

The notion is that we want to locate professionals (people) who work in concert with excellent newspapers rather than simply good newspapers themselves.

The concept is therefore fairly similar to PageRank in that links will be counted as votes.

- You must calculate the hub and authority score for each page.
- Essentially, each page will have a quality score that is unique to it. This is known as a hub score.

- We refer to the quality score each page receives as a content provider also known as its authority score as well.
- The hub score, therefore, would be just the total of all the votes cast for the authority that are referenced. The total number of expert votes that the page receives will be the authority score.
- Next, we will calculate the steady state score using the repeated improvement concept.

PageRank vs. HITS

PageRank	HITS
Computed for all web pages stored prior to the query	Performed on the subset generated by each query
Computes authorities only	Computes authorities and hubs
Fast to compute	Easy to compute, real
Time execution is easy	Time execution is hard

PageRank is a ranking of all web pages based on where they are in the structure of the web graph. PageRank uses backlinks, which are links to other pages on the same site. Backlinks from popular pages aren't as important as backlinks from important pages.

Advantages of Page Rank

- Fighting spam. A page is important if other important pages link to it.

Since it is hard for the owner of a Web page to get links from other important pages to his or her page, it is hard to change Page Rank.

- Page Rank is a global measure that doesn't depend on the search query.

All of the pages' Page Rank values are calculated and stored offline, not when a query is made.

2.5 SIMILARITY

1.5.1 Similarity of Documents

Textually similar documents in a large corpus like the Web or a collection of news articles are easy to find with Jaccard similarity. This is an important class of problems that Jaccard similarity solves well. We should know that the level of similarity we are looking for is character-level similarity, not "similar meaning," which requires us to look at the words in the documents and how they are used. This problem is also interesting, but there are other ways to solve it. But textual similarity is also useful in

important ways. A lot of them have to do with finding duplicates or close copies. First, let's note that it's easy to tell if two documents are exact copies of each other. All you have to do is compare them character by character, and if there are any differences, they are not the same. In many applications, however, the documents are not the same, but they share a lot of text. Here are just a few: Plagiarism Plagiarized documents give us a chance to see how well we can find textual similarities. Plagiarists can only take parts of a document to use as their own. He might change a few words or the order in which the original sentences are written. Still, the new document may have at least 50% of the original. A sophisticated case of plagiarism can't be found by comparing documents character by character. Page Mirrors It is common for important or popular websites to be hosted on more than one server so that the load can be spread out. Most of the time, the pages on these mirror sites will be very similar, but not the same.

For example, each one might have information about its own host and links to other mirror sites but not to itself. Taking pages from one class and putting them in another is a similar thing. These pages might have class notes, homework, and slides from lectures. From one year to the next, similar pages might change the name of the course, the year, and make small changes. It's important to be able to find these kinds of similar pages because search engines give better results when they don't show two pages that are almost exactly the same on the first page of results. Same Source articles It is common for one reporter to write a news story that gets sent to many newspapers, like through the Associated Press. The newspapers then put the story on their websites. Each newspaper makes small changes to the article. They may take out paragraphs or even add their own material. Most likely, they will put their logo, ads, and links to other articles on their site around the article. But the original article will be the most important part of each newspaper page. News aggregators like GoogleNews try to find all versions of an article so they can show only one. To do this, they have to figure out when two Web pages have similar text but are not the same.

2.5.2 Jaccard Similarity and Jaccard-Bag similarity

Jaccard similarity and Jaccard-Bag similarity are both measures of similarity between two sets of items, but they differ in how they handle repeated items.

Jaccard similarity, also known as Jaccard index, is a measure of similarity between two sets A and B, defined as the size of the intersection divided by the size of the union of the sets:

$J(A, B) = |A \cap B| / |A \cup B|$

Jaccard similarity ranges from 0 to 1, where 0 indicates no similarity and 1 indicates complete similarity.

Jaccard-Bag similarity, on the other hand, is a modified version of Jaccard similarity that into account repeated items in the sets. Jaccard-Bag

similarity is defined as the size of intersection of the multisets (bags) of the sets A and B, divided by the size of the union of the multisets:

$J\_B(A, B) = |A \cap\_bag B| / |A \cup\_bag B|$

where $\cap\_bag$ and $\cup\_bag$ represent the intersection and union of multisets.

Jaccard-Bag similarity ranges from 0 to 1, where 0 indicates no similarity and 1 indicates complete similarity. It is typically used in applications where repeated items are common, such as in text analysis and data mining.

Example 2.5.1: In Fig 2.5.2 we can see two sets S and T. There are three elements in there intersection and total eight elements that appear on both sets S an d T. The SIM(S, T) is 3/8.

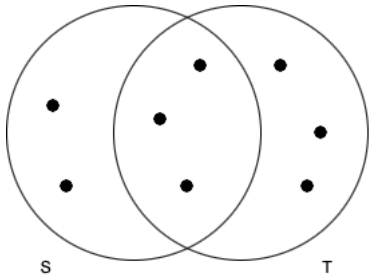

Figure 2.5.2 Two Sets with Jaccard Similarity 3/8

Use Case

Movie Ratings Netflix keeps track of which movies each customer has rented, as well as how they rated those movies. We can think of two movies as similar if many of the same customers rented them or gave them high ratings. We can also think of two customers as similar if they rented or gave high ratings to many of the same movies. The same things we said about Amazon apply here: similarities don't have to be big to be important, and putting movies together by genre will make things easier. Also, the question of ratings adds a new element. There are also:

1. Ignore customers who gave low ratings to the movies they rented. In other words, act as if the customer never rented the movie.

2. To compare customers, think of "liked" and "hated" as two set elements for each movie. If a customer gave a movie a high rating, add a "liked" to the customer's set for that movie. If they gave a movie a low score, put "hated" in their set for that movie. Then, we can look for sets with a high Jaccard similarity. We can use the same method to compare movies.

3. If ratings range from one to five stars, add a movie to a customer's set n times if they gave it n stars. Then, use Jaccard similarity for bags to figure out how much customers are alike. The Jaccard similarity for bags B and C is found by counting an element n times in the intersection if n is the least number of times the element appears in both B and C. In the union, we add up the number of times an element appears in both B and C.

2.5.3 Similarity measure between two vectors

After making the necessary adjustments to the word weights, we now have k-dimensional document and query vectors (where k is number of index terms in vocabulary). So what we need to do is determine what they have in common. The cosine similarity is a commonly used and broadly accepted measure of similarity.

It is necessary to divide the inner product of the two vectors (the total of the components that were multiplied pairwise) by the product of the lengths of the vectors. Because of this, the lengths of the vectors are normalised to be equal to one another, and the resemblance between them can only be explained by the angle, or more specifically the cosine of the angle, that exists between them.

$$sim(d_1, d_2) = \frac{\vec{v}(d_1) \cdot \vec{v}(d_2)}{|\vec{v}(d_1)||\vec{v}(d_2)|}$$

Because of the orthogonality of their vectors, papers that do not share a single word are given a similarity value of zero, while documents that have a comparable vocabulary are given larger values (up to one in the case of identical documents). Since a query may be thought of as a brief document, it is of course feasible to generate a vector for the query. This vector can then be used to compute the cosine similarity between the vectors generated by the query and those generated by the documents that fit the query. The order of the results is determined, in the end, by the similarity values between the query and the documents that were obtained.

There are several ways to measure similarity in Link Analysis and Specialized Search, including:

1. Cosine Similarity: Cosine similarity is a measure of the similarity between two non-zero vectors of an inner product space that measures the cosine of the angle between them. In information retrieval, cosine similarity is often used to determine the similarity between a document and a query.

The cosine similarity between two vectors A and B is defined as:

cosine_similarity(A,B) = (A.B) / (||A|| * ||B||)

where A.B is the dot product of A and B, and ||A|| and ||B|| are the magnitudes of vectors A and B, respectively.

2. Jaccard Similarity: Jaccard similarity is a measure of similarity between two sets of items based on the number of elements that they share. In Link Analysis, Jaccard similarity is used to measure the similarity between two web pages based on their common links.

The Jaccard similarity between two sets A and B is defined as:

$J(A,B) = |A \cap B| / |A \cup B|$

where $|A \cap B|$ is the size of the intersection of sets A and B, and $|A \cup B|$ is the size of the union of sets A and B.

3. PageRank and HITS Algorithms: PageRank and HITS (Hypertext Induced Topic Selection) are two algorithms used to measure the importance of web pages based on their links. PageRank assigns a score to each web page based on the number and quality of links pointing to it, while HITS assigns a score to each page based on its authority and hubness.

The PageRank score of a page i is given by:

$PR(i) = (1-d) + d * \Sigma(PR(j) / C(j))$

where PR(j) is the PageRank score of page j, C(j) is the number of outbound links from page j, and d is a damping factor between 0 and 1 that represents the probability of a user following a link.

The HITS score of a page i is calculated in two steps. First, the authority score A(i) of page i is calculated based on the incoming links to it:

$A(i) = \Sigma H(j)$

where H(j) is the hub score of page j, and j links to page i.

Second, the hub score H(i) of page i is calculated based on the outgoing links from it:

$H(i) = \Sigma A(k)$

where A(k) is the authority score of page k, and i links to page k.

4. Collaborative Filtering: Collaborative filtering is a technique used in recommender systems to predict user preferences based on the past behavior or preferences of other users. There are two main types of collaborative filtering: user-based and item-based.

In user-based collaborative filtering, the similarity between users is calculated based on their past ratings or preferences for items. The predicted rating of an item for a user is then calculated as a weighted average of the ratings of other similar users.

In item-based collaborative filtering, the similarity between items is calculated based on the past ratings or preferences of users. The predicted rating of an item for a user is then calculated as a weighted average of the user's ratings for similar items.

The similarity between two users u and v can be calculated using the cosine similarity measure:

$sim(u,v) = (\Sigma ui * vi) / (\sqrt{\Sigma}ui^2 * \sqrt{\Sigma}vi^2)$

where ui and vi are the ratings of user u and v for a particular item i.

In item-based collaborative filtering, the similarity between two items i and j can also be calculated using the cosine similarity measure:

$$sim(i,j) = (\Sigma u(i)*u(j)) / (\sqrt{\Sigma}u(i)^2 * \sqrt{\Sigma}u(j)^2)$$

where u(i) and u(j) are the ratings of all users for items i and j, respectively.

Based on user evaluations, this similarity metric is used to determine which things are comparable to one another. The system may propose products to a user based on both the highly rated items they have selected and the comparable items that other users have selected highly after calculating the similarity between all pairs of items. A weighted average of the user's ratings for comparable things is then used to get the anticipated rating of an item for a user.

2.6 HADOOP & MAP REDUCE

2.6.1 What is Hadoop?

Hadoop is a framework developed by Apache that is available as open source. It enables users to store and handle large amounts of data in a distributed manner across clusters of computers by using simple programming patterns. It is built with the capability to expand from a single server to thousands of devices, each of which provides local computing and storage.

- Open-source applications A global network of developers produces and updates open-software Although if more and more commercial versions of Hadoop are becoming available, it is free to download, use, and contribute to.
- Framework In this context, it indicates that all the programmes, connections, and other resources you need to create and operate software applications are available.
- Massive storage Big data is divided into units by the Hadoop architecture and stored on clusters of commodity hardware.
- Ability to process for quick results, Hadoop processes massive volumes of data simultaneously on a number of cheap machines.

2.6.2 The benefits of Hadoop

Hadoop's capacity to swiftly store and handle large volumes of any kind of data is one of the main reason’s businesses use it. That's a crucial factor to consider with data quantities and variety continually rising, notably from social media and the Internet of Things. Advantages comprise:

•Computer capability. Big data is processed fast using this concept of distributed computing. The user has greater processing power the more computer nodes they utilise.

•Adaptability. Users do not need to preprocess data before saving it, in contrast to conventional relational databases. They are free to keep as much data as they like, and they may choose afterwards how to utilise it. Data that is not organised, such as text, photos, and videos, is included.

•Tolerant of faults. Processing of data and applications is safeguarded against hardware malfunction. To ensure that distributed computing does not fail, tasks are immediately routed to other nodes if one node goes down. Moreover, all data is automatically stored in numerous copies.

•Low price. The open-source framework is free and uses commodity hardware to store large quantities of data.

• Flexibility. By simply adding new nodes, the user may quickly expand the system. Minimal management is necessary.

Big data

Big data is a collection of expansive datasets that cannot be analysed using conventional computer methods. Big data has evolved into a field in and of itself, including a variety of tools, methodologies, and frameworks.

2.6.3 Ecosystem for Hadoop

Hadoop is a collection of many frameworks and tools for managing, storing, processing efficiently, and analysing large amounts of data. Large datasets may be stored on inexpensive hardware using HDFS, which functions as a distributed file system. The Hadoop resource manager, YARN, manages a cluster of nodes and distributes RAM, memory, and other resources in accordance with the needs of the application.

Data processing is handled by MapReduce, while data transfers between the current Hadoop database and other external databases are handled by Sqoop. Flume for data gathering and digestion, Pig for scripting, Hive for distributed storage querying, Spark for real-time data processing and analysis, Mahout for algorithms, and Apache Ambari for real-time monitoring.

2.6.4 Hadoop's component parts

Hadoop is a system for managing and storing Big Data that makes advantage of distributed storage and parallel computing. It is the programme for handling Big Data that is most often utilised.

Hadoop is composed of three parts.

1. Hadoop HDFS - Hadoop's storage component is the Hadoop Distributed File System (HDFS).

2. Hadoop MapReduce - Hadoop's processing engine is called MapReduce.

3. YARN for Hadoop - A Hadoop resource management component is called YARN.

2.6.5 Applications of Hadoop

Hadoop is now being used in many different industries to meet their unique demands. Yahoo was one of the first businesses to use Hadoop. Since then, a number of well-known companies have incorporated this into their architecture for the benefit of their company, including Facebook, Twitter, and Adobe.

To reduce security concerns in the financial industry, big data in banking and securities may track fraudulent activities, provide early warnings, identify card fraud, audit trails, credit risk monitoring, and manage customer data analytics. Now, the Securities Exchange Commission (SEC) uses network analytics and natural language processing to track and keep an eye on activity.

The Big Data framework in healthcare may assist in a thorough examination of the data present on-site for availability, growing prices, and even monitoring the progression of chronic illness.

Big Data is utilised in the media and entertainment industry to gather, examine, and get meaningful customer information. To further hone corporate processes, it makes use of social media components, media material, and uncovers trends from real-time analytics. Big Data is effectively used during the Grand Slam Wimbledon Tennis Tournament to provide real-time sentiment analysis for TV, mobile, and internet consumers.

The University of Tasmania, an Australian institution, used big data to follow the actions of 26000 individuals and manage their progress in higher education. Similar to this, it was used to evaluate a teacher's performance based on student learning experiences, grades earned, conduct, demographics, and other factors.

Big Data may expand the possibilities of the supply chain in the manufacturing and natural resources sector, increasing productivity. Both industries have a significant quantity of rising volume and velocity untapped data. Incorporating big data technology may increase a system's productivity, dependability, overall quality, and profitability.

Using Big Data frameworks, governments have also optimised a number of processes.

Big Data integration and interoperability often present problems with the public scale. The Food and Drug Administration is currently using big data to examine trends in user behaviour and responses to many factors, including food-related illnesses and disorders.

Hadoop has been used in the transportation industry to manage traffic, develop intelligent transportation systems, design routes, and reduce congestion.

Big Data may be utilised, particularly for the logistics division, to monitor shipments, journey times, and further, conserve gasoline by implementing best practises and giving directions to vehicles.

A more advanced electric grid with smart metres to monitor the reading every 15 minutes will be introduced in Energy and Utilities. To improve system performance, granular data will be used to evaluate data from numerous devices and combine it with user input.

Big Data may monitor customer purchasing patterns in the retail and wholesale industries and compare them to marketing strategies to increase the value of the company. Similar applications include consumer loyalty cards, RFID, point-of-sale scanners, neighbourhood activities, inventory management, and even fraud prevention.

Big Data can monitor consumer insights in the insurance industry to streamline products and forecast behaviour through GPS devices, social media connections, and investment prospects. Claims management may benefit from optimised concepts to provide services more quickly.

2.6.5 Hadoops Architecture

In 2005, Doug Cutting, Mike Cafarella, and their team used the Google solution to launch the HADOOP open-source project, which they named after their son's elephant toy. The Apache Software Foundation has now officially registered Apache Hadoop as a trademark.

Data is processed in parallel on several CPU nodes via the MapReduce algorithm, which is used by Hadoop to execute applications. In summary, the Hadoop framework is capable of supporting the development of applications, allowing them to operate on computer clusters, and doing comprehensive statistical analysis for enormous volumes of data.

The following four modules are part of the Hadoop framework:

• *Hadoop Common*: Other Hadoop modules depend on these Java libraries and tools. The Java files and scripts needed to launch Hadoop are included in these libraries, which also offer abstractions at the OS and filesystem levels.

• *Hadoop YARN* is a framework for managing cluster resources and scheduling jobs.

• HDFSTM (Hadoop Distributed File System): high-throughput access to application data is made possible via a distributed file system.

• *Hadoop MapReduce*: This YARN-based solution is used to handle massive data sets in parallel.

The four parts of the Hadoop framework are shown in the figure below.

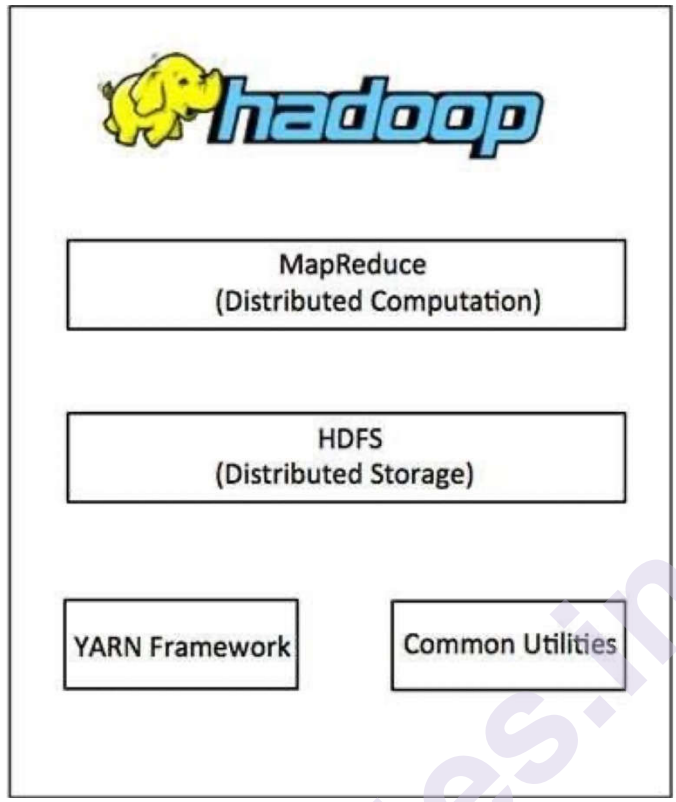

Figure 2.6.1 Hadoops Architecture

Since 2012, the name "Hadoop" has come to refer to a number of other software packages that may be installed on top of or alongside Hadoop, such as Apache Pig, Apache Hive, Apache HBase, Apache Spark, etc. in addition to the foundation modules stated above.

HDFS

Under the conventional method, a single central database housed all of the data. A single database was unable to fulfil the job before the emergence of big data. The large amount of data needed to be stored, thus a distributed method was chosen as the answer. Many distinct databases received the data once it was broken up. A file system called HDFS was created specifically for storing large datasets on common hardware and storing data in diverse formats on many machines.

There are two components in HDFS:

1. The master daemon is named NameNode. There is just one NameNode that is active. It holds all the info and controls the DataNodes.

2. DataNode - The slave daemon is named DataNode. There could be many DataNodes. It keeps the genuine information.

3. We discussed how distributed data storage in HDFS works, but did you realise that the storage system has requirements? Data is divided into numerous blocks by HDFS, with a default block size of 128 MB.

Depending on the distribution of the data and the processing speed, the default block size may be modified.

4. The aforementioned graphic reveals that we have 300 MB of data. There are three parts to this: 128 MB, 128 MB, and 44 MB. It is not necessary to size the last block at 128 MB since it manages the remaining required storage space. This is how data is distributedly stored in HDFS.

5. Now that you are familiar with HDFS, it is important for you to comprehend its foundation and how the HDFS cluster is run. The next thing we're going to look at is something that YARN does.

YARN (Yet Another Resource Negotiator)

YARN, or Yet Another Resource Negotiator, is an acronym. It oversees the node cluster and serves as Hadoop's resource management component. RAM, memory, and other resources are distributed across several programmes through YARN.

YARN consists of two parts:

1. The master daemon is ResourceManager (Master). It controls how resources like CPU, memory, and network bandwidth are distributed.

2. NodeManager (Slave) - The slave daemon that informs the Resource Manager of resource utilisation is called NodeManager.

MapReduce

MapReduce, which distributes vast amounts of data processing in parallel, is the foundation upon which Hadoop data processing is based.

As we can see, processing our large amount of data is necessary in order to provide a result. As a result, the input splits are first formed by splitting up the input data. Data from each split is provided during the first step, called Map, to create output values. The output of the mapping step is collected and organised into blocks of related data during the shuffle and sort phase. The output values from the phase of shuffling are then pooled. Then it gives back a single output value.

In summary, the three parts of Hadoop are HDFS, MapReduce, and YARN.

Hadoop and Map Reduce

Large computer clusters are the target audience for Map Reduce. While hundreds or thousands of computers are accessible in such clusters, individual machines may break at any moment. The goal of a cluster is to solve complex computational problems using inexpensive commodity machines or nodes that are constructed from standard pieces (CPU, memory, disc). Hence, we must break the task into manageable parts that we can allocate and, in the event of failure, reassign in order to achieve robust distributed indexing. The job distribution and reassignment procedure is controlled by a master node.

MapReduce's map and reduce stages divide the computing task into manageable bits for quick processing by conventional processors. Figure 2.6.2 depicts the different MapReduce phases, while Figure below provides an example using a collection of two documents.

Schema of map and reduce functions

map: input → list(k, v)

reduce: (k,list(v)) → output

Instantiation of the schema for index construction

map: web collection → list(termID, docID)

reduce: (⟨termID$_1$, list(docID)⟩, ⟨termID$_2$, list(docID)⟩, …) → (postings_list$_1$, postings_list$_2$, …)

Example for index construction

map: d_2 : C died. d_1 : C came, C c'ed. → (⟨C, d_2⟩, ⟨died,d_2⟩, ⟨C,d_1⟩, ⟨came,d_1⟩, ⟨C,d_1⟩, ⟨c'ed,d_1⟩)

reduce: (⟨C,(d_2,d_1,d_1)⟩,⟨died,(d_2)⟩,⟨came,(d_1)⟩,⟨c'ed,(d_1)⟩) → (⟨C,(d_1:2, d_2:1)⟩,⟨died,(d_2:1)⟩, ⟨came,(d_1:1)⟩, ⟨c'ed,(d_1:1)⟩)

Figure 2.6.2 MapReduce functions. Map generates key-value pairs. Reduce collects all key values into one list. This list is then processed. Index building is shown by the two functions' instantiations. The map phase handles documents distributedly, therefore termID–docID pairings do not need to be sorted properly initially. Terms are shown instead of termIDs for readability[1].

In order to guarantee that the work can be spread fairly and effectively, the input data, in our instance a collection of web pages, are first divided into n splits. For distributed indexing, splits of 16 or 64 MB are suitable. Splits are continuously allocated by the master node rather than being pre-assigned to machines: The next split is allocated to a machine once it completes processing the previous one. The split that a machine is working on is simply transferred to another computer if it crashes or starts to perform slowly due to hardware issues.

In general, by recasting a huge computational issue in terms of manipulating key-value pairs, MapReduce divides it into smaller KEY-VALUE PAIRS portions. A key-value pair for indexing takes the form (termID,docID). The mapping from terms to termIDs is also distributed in distributed indexing, making it more complicated than in single-machine indexing. Maintaining a (perhaps precomputed) mapping for frequent terms that is replicated to all nodes and using terms directly (instead of termIDs) for infrequent words is a straightforward approach.

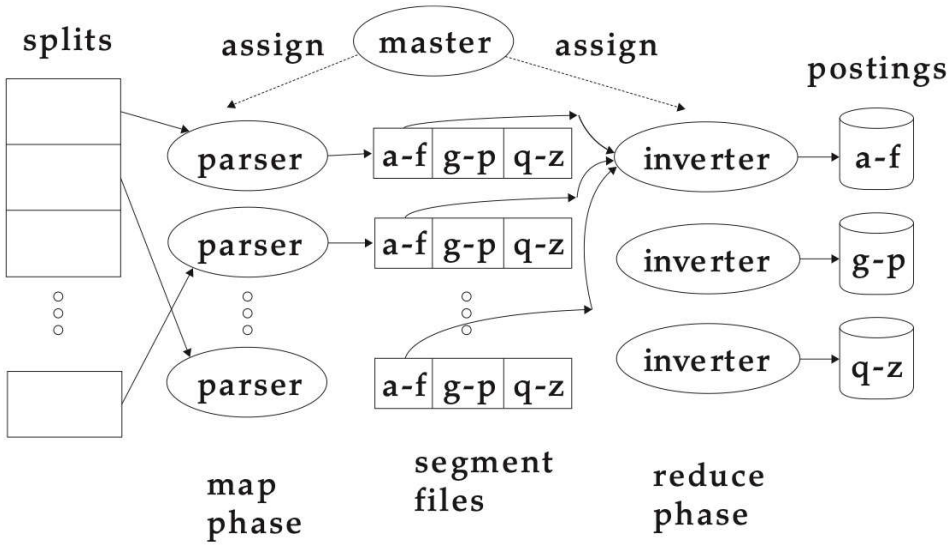

Figure 2.6.3 An example of distributed indexing with MapReduce[1]

We do not address this problem here and assume that all nodes share a consistent

term →termID mapping.

MapReduce's map phase involves splitting up the incoming data into key-value pairs. To local intermediate files, also known as segment files, each parser writes its output (shown as a-f g-p q-z in Figure 2.6.3).

We want to keep all values for a particular key near together in the reduction phase so that they may be read and processed rapidly. The keyvalue pairs for each term partition are written by the parsers into a separate segment file after the keys are divided into j term partitions. The term partitions in the aforementioned figure are listed according to initial letters: a-f, g-p, q-z, and j = 3. The person in charge of the indexing system defines the word "partitions" For each term partition, the parsers then produce associated segment files. This means that each term division corresponds to r segments files, where r is the total number of parsers. For instance, Figure 4.5 depicts the three parsers in Figure 4.6.3 as well as the three a-f segment files of the a-f partition. 2 The inverters' duty during the reduction phase is to compile all values (in this case, docIDs) for a certain key (in this case, termID) into a single INVERTER list. Each term partition is given to a distinct inverter by the master, who also, as with parsers, reassigns term partitions in the event of delayed or failed inverters. One inverter processes each term partition (corresponding to r segment files, one on each parser).

In this case, we suppose that segment files are of a size that is manageable by a single computer. After being sorted for each key, the list of values is then written to the final sorted posts list. Figure 2.6.3 displays the data flow for the values a to f. The inverted index has now been fully built. Inverters and parsers are not two different types of devices. The master recognises idle machines and gives them jobs. In both the map and reduce

phases, the same machine may function as both an inverter and a parser. Moreover, there are often additional tasks that occur concurrently with index building, thus a machine may do crawling or another unrelated work in between being a parser and an inverter.

Each parser writes its segment files to its local disc to decrease writing times before inverters reduce the data. The location of the appropriate segment files is sent by the master to an inverter during the reduction phase (e.g., of the r segment files of the a–f partition). Each segment file only has to be read sequentially once since the parser wrote all information pertinent to a certain inverter to a single segment file. This configuration reduces the amount of network traffic required for indexing.

The overall structure of the MapReduce functions is shown in Figure 2.6.2. Input and output are often listing of key-value pairs so that MapReduce tasks may be executed one after the other. The newly produced term-partitioned index is converted into a document-partitioned index by another MapReduce process. An effective and theoretically straightforward framework for performing index building in a distributed setting is provided by MapReduce. With sufficiently big computer clusters, it may scale to virtually arbitrary huge collections by offering an automated approach for breaking index generation into smaller jobs.

Consider the issue of calculating the number of times each word appears in a large collection of papers.

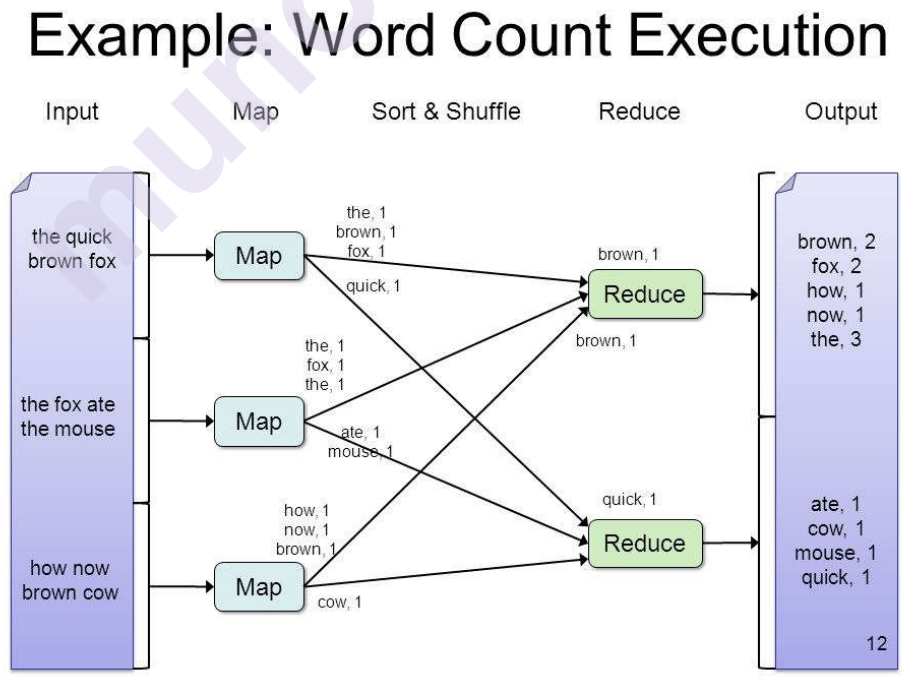

Figure 2.6.4 Word Count Execution

Word Count is a classic example of a problem that can be solved using MapReduce, a programming model for processing large datasets in a distributed computing environment.

In the Word Count problem, the goal is to count the number of occurrences of each word in a large corpus of text. MapReduce is well-suited to this task because it allows the data to be processed in parallel across multiple machines, enabling faster processing of large datasets.

Here is a high-level overview of how Word Count can be executed using MapReduce:

Map phase: In this phase, each input document is broken up into individual words, and a key-value pair is created for each word. The key is the word itself, and the value is 1, indicating that the word has been observed once. For example, the text "the quick brown fox jumps over the lazy dog" would be mapped to the following key-value pairs:

("the", 1)

("quick", 1)

("brown", 1)

("fox", 1)

("jumps", 1)

("over", 1)

("the", 1)

("lazy", 1)

("dog", 1)

Shuffle and sort phase: In this phase, the key-value pairs are shuffled and sorted by key, so that all pairs with the same key are grouped together. This allows the next phase to efficiently count the number of occurrences of each word.

Reduce phase: In this phase, the key-value pairs are processed in parallel by multiple reduce tasks. Each task takes a key and a list of values as input and outputs a key-value pair representing the count of occurrences for that key. In the case of Word Count, the value list for each key consists of a sequence of 1s, indicating the number of times that word appears in the input documents. The reduce task then sums up the values and outputs a key-value pair with the word as the key and the count as the value. For example, the output for the previous example input would be:

("the", 2)

("quick", 1)

("brown", 1)

("fox", 1)

("jumps", 1)

("over", 1)

("lazy", 1)

("dog", 1)

Final output: The output from the reduce tasks is combined to produce the final output of the Word Count program, which is a list of key-value pairs representing the word and its count. This output can be further processed or stored for later use. Overall, MapReduce provides an efficient and scalable approach to solving the Word Count problem, and can be applied to a wide range of other data processing tasks as well.

2.7 PERSONALIZED SEARCH

Finding what one is seeking for becomes more difficult when there is an increasing amount of information. While a remedy for this issue has not yet been developed, personalising online search could be a smart idea. For instance, a person looking for "apple" may be seeking for recipes and culinary advice. The search results might be readily improved by categorization, or perhaps the person has lately been browsing for recipes. The search history in this situation, or even the browsing history, might be considered. To customise search results, there are several methods. Making a user profile is a fundamental and popular concept.

2.7.1 Collecting information

There are several options available for gathering user input data. The user's prior search requests or the search results they clicked may be gathered. Moreover, a source of data is the browsing history. For every page that has been previously visited, a wealth of information may be acquired, including the date and time of the visit, the previous and total number of visits, the page's title or text content, the user's number of out-links followed, and more. The user's bookmarks might also be examined. Data about the customer might also be a source: The access mode (desktop or mobile device), the IP address (the location may be discovered from the IP address), or the user's browser and operating system can all be identified.

2.7.2 Creating profiles

User profiles are really created in two steps. Before creating the profiles in the first section, some pre-processing must be done: The data must first be screened and cleaned up (elimination of noise). Moreover, it is necessary to guarantee user and session identification. The data is refined in the second section using Pattern Discovery to create sample user profiles. Using statistical and machine learning methods, association discovery, and sequential pattern discovery to the data reveals information about the user. Text clustering is one method for generating surfing patterns using data gathered from websites visited. A profile may also be created by categorising the individual. There are several ways to do this. For instance,

the Open Directory Project allows users to search up websites from their history.

The **Open Directory Project** (ODP), usually referred to as DMOZ, is a manually curated directory of websites. DMOZ stands for "Directory Mozilla."

The ODP does not rate or promote websites; its goal is to list and classify them.

Pattern Discoverythe pattern that was utilised to extract a word or phrase from the text. The patterns that may be gleaned from a written document are many, but not all of them are fascinating. Only those considered to be fascinating in some way are considered to be helpful information. The usage of decision trees is another option. To further hone the profile, certain Post-Processing (such as assessments) may be used.

2.7.3 Profile representation

After the actual profile creation, the extracted information has to be stored somehow. For Personalized Web Search the extracted knowledge is incorporated in a Personalization module in order to facilitate the personalization functions. Many different forms of data representation can be applied here. The two most common representations seem to be the rather simple “list of URLs” and the “bag of words”. Different tables or tensors can be used as well (for example: all queries by a user and/or clicked results for every query).

The use of classification is also quite common. The categories for the classification can be taken from the Open Directory Project (or from the Yahoo! Directory), selected from another given hierarchy or self-defined.

2.8 COLLABORATIVE FILTERING AND CONTENT-BASED RECOMMENDATION OF DOCUMENTS AND PRODUCTS

Collaborative filtering is a category of applications where similarity of sets is crucial. In this approach, we propose to customers products that other users who have shown similar likes have enjoyed. Millions of people use Amazon.com, which also offers millions of products. The database keeps track of which clients purchased which things. If two consumers' collections of bought things have a high Jaccard similarity, we may claim that they are comparable. Likewise, two goods will be considered similar if their sets of buyers have significant Jaccard similarity. While we could anticipate mirror sites to have Jaccard similarity exceeding 90%, it is doubtful that any two clients would have that high of a level of similarity (unless theyhavepurchased only one item). Even a Jaccard similarity of 20% can be exceptional enough to locate clients with comparable interests. The same is true for things; Jaccard similarities do not necessarily need to be high to be meaningful. In addition to identifying related clients or products, collaborative filtering necessitates a number of

technologies. For instance, two science fiction book fans who shop on Amazon could each purchase a large number of titles, but they will only have a small number of titles in common. But, by combining similarityfinding with clustering, we may be able to identify the similarities across science fiction literature and group them together. Next, by determining whether they made purchases inside a large number of the same groupings, we may develop a more potent idea of customer similarity.

2.8.1 What is CF?

In a nutshell, it is a "word of mouth" endorsement. "The process in which the consumer of a product or service informs friends, relatives, neighbours, and acquaintances about its qualities," according to the Oxford Dictionary, "particularly when this occurs prior to media promotion."

As an example, let's say that I read a book, like it, and told my friends about it on social media. My friends who like reading novels similarly to me could opt to read the book and then suggest it to their friends.

This procedure may be automated as follows on an e-commerce website. An implicit suggestion occurs when I purchase a book, but the website may ask me to explicitly rate the book, say on a scale of 1 to 10.

The CF system will be able to determine that my buddy has similar reading preferences to mine when he goes onto the website since we have already made comparable purchases. The computer will also detect that he has not yet purchased the book I highly recommended, and will then suggest this book to my buddy. The core of collaborative filtering is this.

Before showing the user the top suggestions, the algorithm will gather as many recommendations as it can and rank them based on their general popularity. CF may be used in online navigation to suggest connections to web users who have similar interests, as well as in e-learning to propose information to instructors and students.

a) User-Based Collaborative Filtering

	Item				
User	Data Mining	Search Engines	Databases	XML	...
Alex	1		5	4	...
George	2	3	4		...
Mark	4	5		2	...
Peter			4	5	...

Table 1.8.1 User -Item Rating Matrix

Have a look at the user-item matrix in Table 2.8.3. Each user is represented by a row, and each item by a column. The rating that user I gave to item j is represented by a number in the ith row and jth column; an

empty cell means the user did not rate that item. The ellipsis (...) at the end of each row denotes the fact that we have only shown a portion of the available entries.

When an item hasn't been reviewed yet, one may wonder how it may be endorsed. An e-commerce site could nevertheless wish to highlight products with no ratings, in which case a content-based strategy is required.

b) Item-based Collaborative filtering

Instead of matching users or customers who have similar tastes in rated things, item-to-item recommendation systems aim to pair comparable items that have been co-rated by various people. With user-based techniques, item-to-item CF examines column similarity rather than row similarity.

c) Model based collaborative filtering

There are also alternative approaches that create a statistical model of the user-item matrix using machine learning techniques and utilise that model to generate predictions. One such method develops a neural network that learns to anticipate each user's rating of a new item. Another method creates association criteria such as "30% of users like all these products, and 90% of users who like items I and j also like item k."

The rules are generally of the form X ⇒ Y, where X is a set of items and Y is another item, as in user-based algorithms. In this case, the rule is {i, j} ⇒ {k}. According to the rule, 30% of the users in the user-item matrix support it, meaning that they all like all three products (this includes the items in both X and Y). The 90% is the confidence in the rule, which is the percentage of people who like all three things (including those in either X or Y) compared to those who just like items I and j. (this includes only the items in X).

Another method employs the naive Bayes classifier with the user-item matrix as the input and states that an item is liked by the active user if other user ratings are considered, with the active user's liking of the item being proportional to the product of the probabilities that each other user will also find the item to be liked.

2.8.2 Content-based recommendation of documents

Under the framework of user-based collaborative filtering, there are two issues. Since a user will often only rate (or buy) a small number of things, say 30, out of the millions that may be offered, the user–item matrix is quite sparse. When an item has not yet been evaluated, there is another difficulty that is similar to this one called the first-rater dilemma. A content-based strategy is essential in this scenario since an e-commerce website could still wish to promote products that do not have any ratings associated with them. In order for content-based systems to be successful, the system must have the capability of compiling an interest profile for each user.

The user's preferred categories in regard to the application are considered to be part of the user's interests. Take, for instance, whether the user favorsworks of fiction over works of nonfiction, or mainstream music over classical music. Once the system has a user profile, it is able to examine the degree to which the item (or content) a user is seeing is similar to the profile, and it is then able to establish a rating for the item based on the degree to which it is similar (or content). This is quite similar to the search procedure, except that the user's profile will serve as the query, and the things that will be shown to the user will serve as the results of the query. When an item has a higher aggregate rating, the user will see it ranked higher overall when they see it.

2.9 HANDLING"INVISIBLE" WEB

The Surface Web is what we often access using well-known search engines like Google, Yahoo, or Bing. These well-known search engines sift through tens of billions of pages of material and provide it to us as needed. Yet this is only the very top of the iceberg. The Deep or Invisible Web is what is considered to be underneath the Surface Web. It consists of: • Private websites, such VPNs (Virtual Private Networks) and login-required websites

• Sites with limited access to content (which restrict access in a technical way, such as by using Captcha, Robots Exclusion Standard, or no-cache HTTP headers that prevent search engines from browsing or caching them); • Sites with unlinked content (which lacks hyperlinks to other pages), which prevents web crawlers from accessing information; • Text that is frequently encoded in image or video files or in particular file formats that search engines do not support; • Dynamic content made specifically for a single

• Content that is downloaded using Flash and Ajax solutions, scripted material, and sites that can only be accessed via Java Script.

Inside the unseen web, there are a lot of high-value collections to be discovered. There, you may discover information that most people would be familiar with and perhaps find beneficial, such as:

• Academic studies and papers

• Blog platforms

• Pages created but not yet published

• Scientific research

• Academic and corporate databases

• Government publications

• Electronic books

• Bulletin boards

• Mailing lists

• Online card catalogs

• Directories

• Many subscriptions journal

• Archived videos

• Images

2.9.1 Open Access Journal Databases

Open access journal databases (OAJD) are collections of open access academic publications that are kept in a way that makes them accessible to scholars and other people looking for a particular piece of knowledge or information. These databases are found in the invisible web since they are made up of unconnected material. Databases like "AGRIS" (International Information System for Agricultural Science and Technology), "BioMed Central," "Copernicus Publications," "Directory of Open Access Journals," etc. are examples of well-known and respectable databases.

2.9.2 Invisible Web Search Engines

There are major differences in the search engines that provide results from the unseen web. These deep web engines are more focused and often only access one kind of data. This is because each sort of data has the potential to provide an astronomical number of outcomes. A broad deep web search would soon become like looking for a needle in a haystack. Deep web searches are thus more deliberate in their initial query specifications. Popular invisible online search engines are listed below:

• The meta search engine "Clusty" not only aggregates information from several sources but also generates "clustered" results that are automatically sorted by category.

• The invisible web's "Complete Planet" searches more than 70,000 databases and specialised search engines. a search engine that works equally effectively for academics and casual users.

• A real librarian manages "Digital Librarian," which is A Librarian's Pick of the Best on the Web. Digital Librarian includes information from areas as varied as Activism/Non-Profits and Railroads and Waterways, with an eclectic mix of around 45 major categories.

• The Regents of the University of California's "Info Mine" is yet another collection of online resources created by librarians.

• The "Wayback Machine," which enables users to find archived papers, is only one of several categories on "Internet Archive," which also offers an archive of Grateful Dead soundboard and crowd recordings. They provide 126K live music performances, 6 million texts, 2.5 million movies, and 2.9 million audio recordings.

• The Drexel University website "The Internet Public Library" (ipl1 and ipl2) is a non-profit organisation maintained by students. Students voluntarily take on the role of librarians and assist guests with their inquiries. Data categories that are geared for kids and teens are among them.

2.9.3Ways to Make Content More Visible

The Search Engine Optimization (SEO) sector was born out of the notion of making information more visible. Among the techniques to raise your search engine optimisation are:

• Sort your database by category. If you have a database of goods, you might publish a subset of the data on static overview and category pages to make material accessible without a form or query. Job listings are a good example of information that benefits from this.

• Create internal connections on your website, tying your own pages together. Spiders will index each link, increasing the visibility of your website.

• Make a sitemap public. It's essential to provide a fresh, serially linked sitemap to your website. It's no longer regarded as best practise to advertise it to your visitors; instead, you should post it and maintain its current so that spiders can evaluate your site's content effectively.

• Write somewhere about it. Finding methods to provide connections to your website on other websites is one of the simplest SEO techniques. It will become more noticeable as a result.

• Promote your website via social media. On Twitter, Instagram, Facebook, or any other social media network that fits you, provide a link to your website. You'll boost the number of links on the Internet and increase traffic to your website.

• Eliminate access limitations. Avoid requiring logins or time limits unless you are trying to sell subscriptions.

• Produce logical code. Verify the code of your website so that spiders can simply browse it, even if you use a pre-packaged website design without changing it.

• Be sure to pay attention to keywords that are pertinent to your content when creating page titles and link names for your website.

2.9.4 How to Access and Search for Invisible Content

If a site is inaccessible by conventional means, there are still ways to access the content, if not the actual pages. For invisible content that cannot or should not be visible, there are still a number of ways to get access:

• Join a professional or research association that provides access to records, research and peer-reviewed journals.

• Access a virtual private network via an employer.

• Request access; this could be as simple as a free registration.

• Pay for a subscription.

• Use a suitable resource. Use an invisible Web directory, portal or specialized search engine such as Google Book Search, Librarian's Internet Index, or BrightPlanet's Complete Planet.

2.9.5 Invisible Web Search Tools

Below is a short selection of tools (directories, portals, search engines) for finding invisible online material.

• Art - Louvre Museum

• The Online Books Page for Books Online

• FreeLunch.com's Economic and Employment Statistics

• Bankrate.com's finance and investing section

• GPO's Catalog of US Government Publications for general research

Government Data - Copyright Records (LOCIS); International - International Data Base (IDB); Library of Congress - Library of Congress; Law and Politics - THOMAS; PubMed; Transportation - FAA Flight Delay Information;

2.10 SNIPPET GENERATION

Snippet, a concise summary of the content that is intended to let the user determine its applicability. The title of the document plus an automatically derived brief summary often make up the snippet.[2]

Figure 2.10.1 The first 4 snippets from Google for German Expressionism Brucke

For instance, the figure above provides some sample snippets from Google that provide a summary of the first four articles that were retrieved in response to the query "German Romanticism Brucke."

Figure 2.10.2 Example of product reviews rich snippets

A snippet is a brief summary or excerpt of a document, such as a web page or article, that is intended to give users an idea of the content and help them determine its relevance or applicability to their needs. The snippet typically includes the title of the document and a brief summary of its content, which may be automatically generated by search engines or other software. Snippets are commonly used in search results, where they provide users with a quick overview of the content and help them decide which results to click on for more information.

2.11 SUMMARIZATION, QUESTION ANSWERING

A user's requirement for information represented in a natural language inquiry is often satisfied by the precise words, sentences, or brief passages produced by question answering (QA) and summarization activities. We often prefer to provide a single clear, succinct response rather to requiring the consumer to wade through a lengthy document. Dealing with factual inquiries is the simplest kind of question answering. Simple facts that can be obtained in brief text strings are the answers to factoids. The examples of this kind include the following.

•Who was the Virgin Airlines founder?

• At what age does autism often manifest?

• What city is home to Apple Computer?

A text string containing a person's name, a temporal expression, or a place may each be used as an immediate response to one of these inquiries. Hence, factoid inquiries are those whose solutions can be found in brief

passages of text and which belong to a particular, readily defined category, often a named item of the like.

These solutions may be found in a smaller literature collection or alternatively on the Internet. Sometimes we need knowledge that covers more ground than a single statistic, but not as much as a whole document. In certain situations, a summary of a document or series of papers may be necessary.

Text summary aims to create an edited version of a text that includes the crucial or pertinent information. The two fundamental types of summaries are dynamic (or query-dependent), which are tailored based on the user's information needs as inferred from a query, and static, which are always the same regardless of the question.

A static summary often consists of either or both a subset of the original content and any accompanying information. The most basic kind of summary pulls the first two lines or the first 50 words of a document, or it focuses on a specific area of the text, such the title or author.

The goal of dynamic summaries is to offer the parts of the document that will be most useful to the user in assessing it in light of their information needs. They may display one or more "windows" on the page. These windows are sometimes referred to as keyword-in-context () snippets since they typically include one or more of the search keywords.

Current study focuses on many significant types of summaries, such as: outlines of any document, abstracts of scientific articles, headlines of news articles, and snippets summarising web pages on search engine results pages. A (spoken) business meeting's action items or other summaries. Email thread summaries.

• solutions to complicated inquiries that are created by summarising many documents; compressed phrases for creating simpler or compressed language;

These summary objectives are often distinguished by their location on the two-dimensional plane.

• Summarizing a single document as opposed to many documents

• Generic versus query-specific summarization

A general summary is one in which the main points of the material are presented without taking into account the needs of any individual users or information seekers (s).

The summary is created in response to a user query in query-focused summarising, also known as focused summarization, topic-based summarization, and user-focused summarization. A lengthy, non-factoid response to a user inquiry might be considered query-focused summarization.

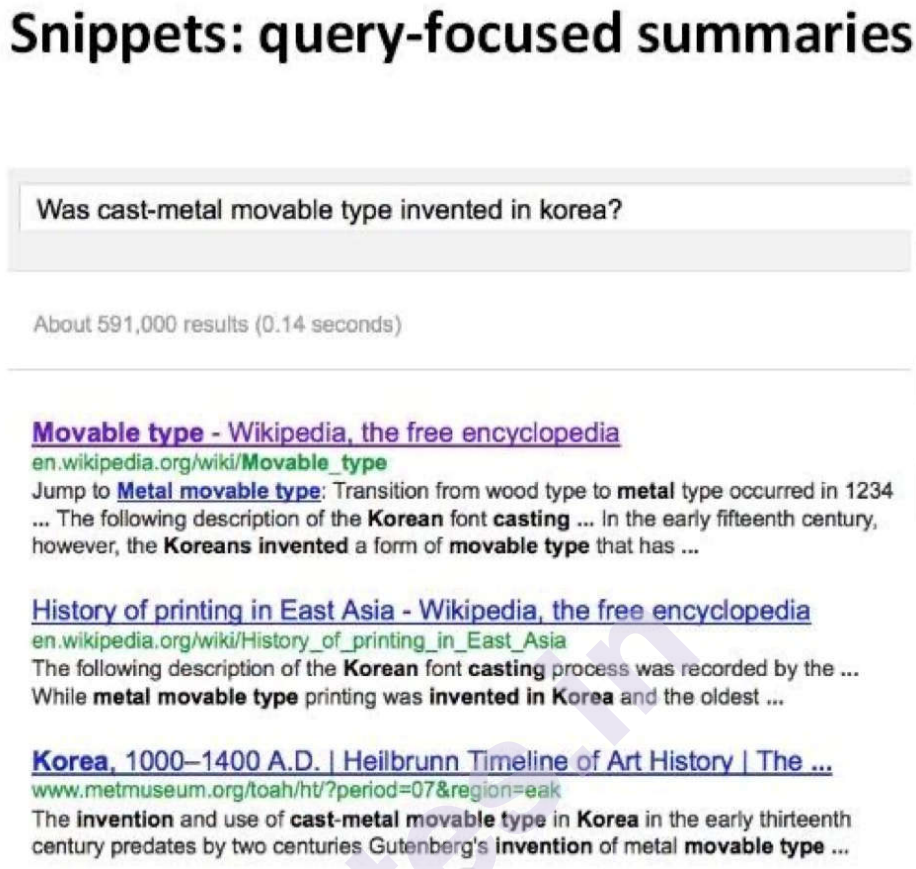

Figure 2.12.1 Query focus summaries

2.12.1Summarizing Single Documents

We are provided a single document to summarise in single document summarization. Hence, single document summarising is used when the objective is to describe the content of a single document, such as when creating a headline or an overview.

Think about creating an extracting summary for only one document. assuming that the extracted units are at the sentence-level. For this assignment, there are three steps of summarization:

2. Content selection: Choose phrases from the text to extract

2. Information Sequencing: Choose a sequence for these phrases in the summary.

3.Sentence Realization: Clean up the sentences by, for instance, getting rid of words that aren't necessary, combining many sentences into one, or correcting issues with coherence.

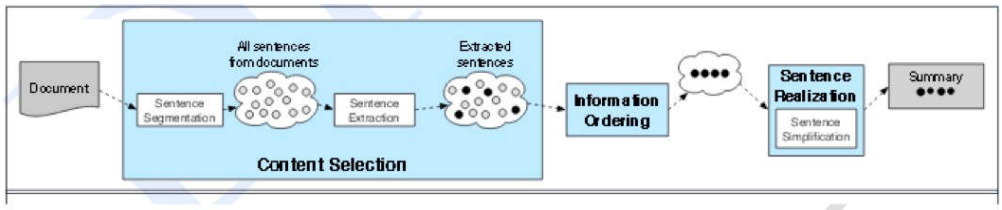

Figure 2.12.2 The basic architecture of a generic single document summarizer

2.12.2 Summarizing Multiple Documents

A collection of documents are the input for multiple document summary, and our objective is to provide a condensation of the full group's information. When we need to synthesise and compress online resources on the same subject, or when we have numerous documents on the same issue, we may utilise multiple document summarization.

The same three processes we've seen previously serve as the foundation for multi-document summarising algorithms. We often think that we must first choose the content, organise the information, and create the sentences from a group of papers that we want to describe.

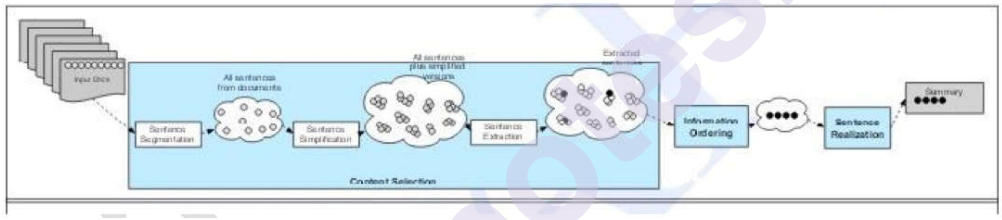

Figure 2.12.3 The basic architecture of a multi-document summarize

2.12 CROSS-LINGUAL RETRIEVAL

A branch of information retrieval called cross-linguistic information retrieval (CLIR) focuses onfinding information that was written in a language other than the user's query language. For instance, a user may enter a query in English and get results for relevant materials in French. The majority of CLIR systems use translation methods to do this.

Based on various translation resources, CLIR approaches may be divided into many categories:

• Machine translation-based CLIR approaches;

• Parallel corpora-based CLIR techniques;

• Comparable corpora-based CLIR techniques;

2.14.1 Challenges in CLIR

The following obstacles must be overcome in order to build a CLIR system:

1. Translation ambiguity: During the process of translating from one language to another, many translations may be available. The choice of an

adequate translation is difficult. • The term "man," for instance, may signify either "neck" or "respect."

2. Identification and translation of phrases • It might be challenging to identify sentences in a constrained context and translate them as a whole rather than word by word.

3. Translate or transliterate a term: • Certain names are unclear; therefore, transliteration is required rather than translation. • For instance, the Marathi word for "sun" (Bhaskar) also denotes a person's name. It might be difficult to identify these circumstances solely on the given context.

4. Mis transcribed words:

Mistakes in transliteration may result in the erroneous term being retrieved in the target language. 5. Dictionary coverage the comprehensiveness of the dictionary is a crucial performance factor for translations utilising a bilingual dictionary.

5. Font: • A large number of online publications are not in Unicode format. To be processed and stored later, these documents must be translated to Unicode format.

6. Morphological examination (different for different languages)

7. Problems with vocabulary (OOV)

• New terms are introduced to the language that the machine may not understand.

2.13 LET US SUM UP

Link analysis and specialised search are two subjects that this section discusses in great detail. In order to discover significant sites inside a network, it presents methods like PageRank and HITS and discusses the significance of link analysis. The section also discusses content-based recommendation systems, collaborative filtering, and similarity metrics that may be used to tailor search results. It is suggested that Hadoop & MapReduce be used as a method for effectively processing massive volumes of data concurrently. Together with strategies for managing the "invisible" web, creating snippets and summaries, and cross-lingual retrieval, evaluation measures are also covered. The hubs and authority's algorithm, which may be used to calculate PageRank, and link spam are discussed as the unit ends. The different methods and algorithms used in link analysis and specialised search are introduced in detail throughout this unit.

2.14 LIST OF REFERENCES

[1] Introduction to Information Retrieval, C. Manning, P. Raghavan, and H. Schütze, Cambridge University Press, 2008

[2] Modern Information Retrieval: The Concepts and Technology behind Search, Ricardo Baeza

-Yates and Berthier Ribeiro - Neto, 2"' Edition, ACM Press Books 2012.

[3] Search Engines: Information Retrieval in Practice, Bruce Croft, Donald Metzler and TrevorStrohman, 1ª Edition, Pearson, 2009.

[4] Information Retrieval Implementing and Evaluating Search Engines, Stefan Büttcher, Charles L. A. Clarke and Gordon V. Cormack, The MIT Press; Reprint edition (February 12, 2016)

2.15 BIBLIOGRAPHY

[1] Manning, C., Raghavan, P., &Schütze, H. (2008). Introduction to Information Retrieval. Cambridge University Press.

[2] Baeza-Yates, R., & Ribeiro-Neto, B. (2011). Modern Information Retrieval: The Concepts and Technology behind Search (2nd ed.). ACM Press Books.

[3] Croft, B., Metzler, D., &Strohman, T. (2009). Search Engines: Information Retrieval in Practice (1st ed.). Pearson.

[4] Büttcher, S., Clarke, C. L. A., & Cormack, G. V. (2016). Information Retrieval Implementing and Evaluating Search Engines. The MIT Press; Reprint edition.

2.16 UNIT END EXERCISES

1. i) Compute the Jaccard similarities of each pair of the following three sets:

   {1,2,3,4}, {2,3,5,7} and {2,4,6}

   ii) Compute the Jaccardsimilarities of each pair of the following three sets:

   {1,1,1,2}, {1,1,2,2,3} and {1,2,3,4}

   iii) Compute the Jaccard similarities of each pair of the following two sets:

   {a,a,a,b} and {a,a,b,b,c}

2. What is the basic principle of flow model in PageRank?
3. How can you represent webpages and their links?
4. What are the issues associated with the web structure?
5. Define Collaborative filtering as a similar sets problem with examples.
6. What is similarity of documents. Give examples.
7. Define Jaccard Similarity of sets.
8. Briefly explain Hadoop MapReduce. Give example. Explain with diagram.

❖❖❖❖

3

WEB SEARCH ENGINE

Unit Structure

3.0 OBJECTIVES

- To understand the fundamentals of web search
- To get familiar with various concepts, tools and techniques for managing the web search

3.1 INTRODUCTION AND OVERVIEW OF WEB SEARCH

A tool used to find information on the World Wide Web is known as a search engine. The search engine enables users to request information that meets particular criteria (usually those that contain a particular word or phrase), and it then returns a list of references that satisfy those requirements.

Large-scale full-text web page indexes are essentially what search engines are. To work swiftly and effectively, they use indexes that are often updated. The quality of search results is determined by the quality of the indexes and how the engines use the data they have. The back of the book indexes is comparable, but search engine indexes are much more complicated. The searching abilities can be much enhanced by understanding even a little bit about how they are made and employed.

Additional types of search engines include mobile search engines, personal search engines that search on individual computers, and enterprise search engines that search intranets. Several search engines also mine information from massive databases, open directories like DMOZ.org, newsgroups, and other sources. Search engines use algorithms rather than human editors to maintain Web directories. The majority of websites that identify as search engines are actually only front ends for search engines that belong to other businesses.

3.2 WEB ENGINE STRUCTURE/ ARCHITECTURE

Following are 4 of the basic components of a search engine:

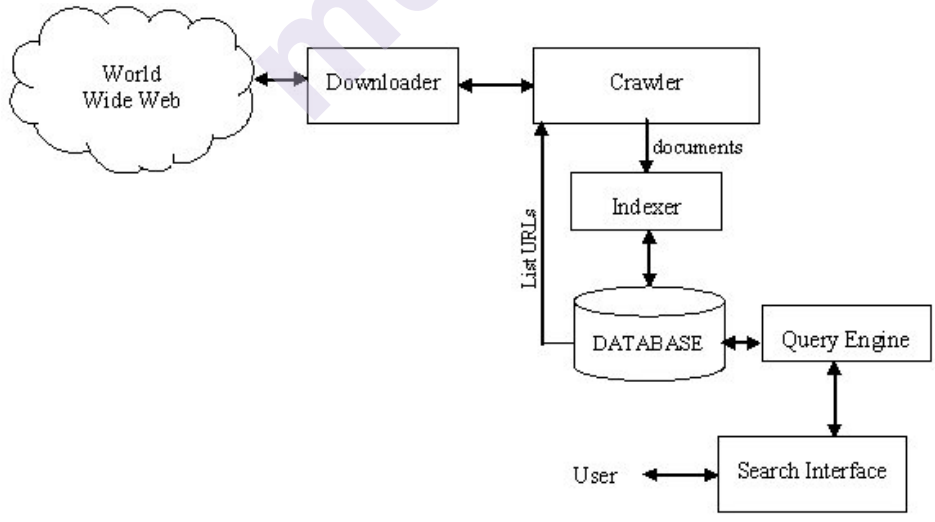

Figure 1: Web engine structure/ architecture

1] Web crawler

Search engine bots, web robots, and spiders are other names for web crawlers. In search engine optimization (SEO) strategy, it is crucial. It mostly consists of a piece of software that browses the web, downloads information, and then gathers it all.

There are the following web crawler features that can affect the search results

- Included Pages
- Excluded Pages
- Document Types
- Frequency of Crawling

2] Database

An example of a non-relational database is the search engine database. That is where all of the data on the web is kept. It has a lot of online resources. Amazon Elastic Search Service and Splunk are two of the most well-known search engine databases.

The following two characteristics of database variables may have an impact on search results:

- Dimensions of the database
- The database's recentness

3] Search interfaces

One of the most crucial elements of a search engine is the search interface. It serves as the user's interface with the database. In essence, it aids users with database query searches.

There are the following features Search Interfaces that affect the search results -

- Operators
- Phrase Searching
- Truncation

4] Ranking algorithms

Google uses the ranking algorithm to determine the order of web sites in its search results.

The following ranking factors have an impact on the search results:

- Location and frequency
- Link Evaluation
- Clickthrough analysis

3.3 THE USER INTERFACES

The query interface and the answer interface make up the user interface of search engines. The fundamental query interface consists of a box into which a string of text is typed. Different search engines return various results depending on the order of the words. For instance, HotBot conducts a search by the intersection of these terms, but AltaVista conducts a search by the union of these terms (all terms must occur in the documents returned as results). Some search engines have a sophisticated query interface that supports Boolean operators as well as additional capabilities including phrase, proximity, URL, title, date range, and data type searches.

Search engines typically return sites in the order of relevancy to the question regarding the answer interface. In other words, the sites that are most pertinent to the search are listed first. Every entry in the list of results typically has the title of the page, a URL, a succinct synopsis, a size, a date, and a written language.

3.4 PAID PLACEMENT

A Web search engine's relevant search results are displayed alongside advertisements in a pay for placement, or P4P, Internet advertising model. In this strategy, advertisers participate in an open auction to bid for the opportunity to display an advertisement that contains particular search terms (also known as keywords).

A set of results are commonly returned by internet search engines after processing a user's search query against a database index, usually in descending order of relevance score. The user chooses particular content providers from the list for additional transactions based on these results. It is commonly established that there is a high correlation between a result term's ranking and the likelihood that the user would click the result. The possibility that a search engine user will transact with a commercial content source is what commercial content producers refer to as clickthroughs and conversion rates. As a result, content providers are motivated to pay the search engine in order to be included, rated highly, or prominently displayed in the search result, which is known as sponsored placement.

For specific search phrases, this might really include a higher relevance score, a prominent listing, or even a guarantee of retrieval. A screenshot from a comparative shopping engine, seen in Figure 2, shows paid placement (featured listings), standard placement, and advertising.

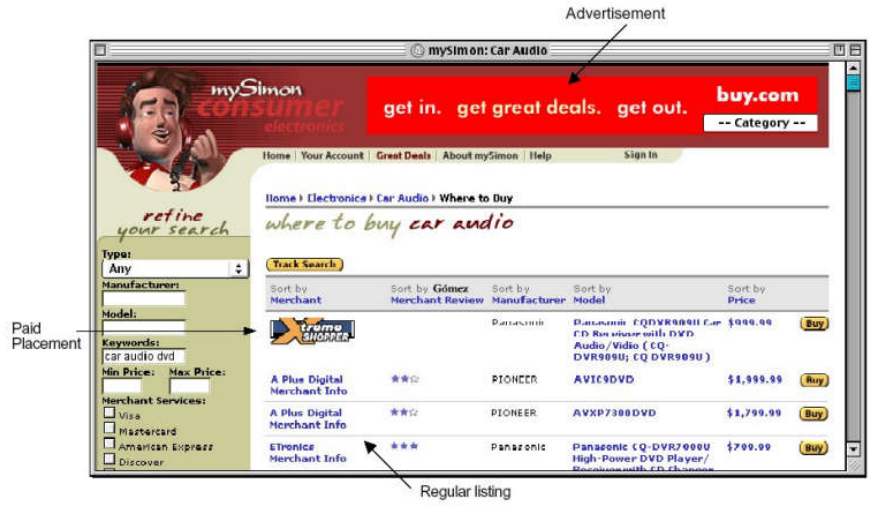

Figure 2: Paid placement, regular listings, and advertising in a comparison-shopping engine. The top listing and graphical icon increase the likelihood that the paid placement listing will be followed up.

On the one hand, paid placement seems to be a need for businesses, and most significant Web search engines accept it. Paid placement, on the other hand, may reduce the search engine's market share and its potential for user-generated revenue.

3.5 SEARCH ENGINE SPAM

It is simple to avoid purchasing Spam if you are supermarket shopping. It's a different story in the world of search engine optimization, though. Every day, we get thousands of "spam" emails. Spamming the search engines is another definition of spam. When working on any search engine optimization for a website, you should try to avoid it at all costs.

What is search engine spam?

Any method used to deceive or "spoof" search engine robots or spiders is referred to as "spam" or "spamming the search engines." Webmasters and website owners are always on the lookout for innovative strategies to deceive the search engines into giving their website a higher rating. Search engine masking is one such technique. On the opposing side of the conflict, programmers, administrators, and engineers working for search engines themselves are constantly modifying their definitions of spam in an effort to penalise websites that employ spam strategies. Because they strive to deliver the most relevant results to their consumers, search engines despise spam because it devalues the value of their findings.

You can unintentionally be employing spam if you are not up to date on the most recent search engine standards regarding what counts as spam. That alone is a very compelling justification for working with a search engine optimization expert like Metamend to maintain your website abreast of the most recent guidelines and algorithms.

We've listed a few of the most popular spam techniques below, along with an explanation of their classification, to assist you get a better picture of what they entail. Hopefully, this has given you some insight into where the next change might come from as well as some understanding of what search engines are looking for.

- **Metatag stuffing**

Webmasters used to be able to include as many keywords as they wanted in their meta tags. A sufficient number of instances of the word "radioactive" in your metatags essentially guaranteed a top position for your site for that keyword because these metatags were the major technique utilised by most engines to evaluate a web site. Search engine administrators-imposed character restrictions for the meta description and meta keywords elements in response to the stuffing problem. This implied that they would only read the first "X" characters or that the tag you were using would be completely ignored if it exceeded the character limit. This meant that the site would no longer greatly benefit from repetitive phrases.Eventually, the search engines put a limit on the number of times a term may be used within a tag to counteract websites that just repeated the term "travel" 50 times and ranked highly for that one term.

They may "spam" your site out of their index if you over that limit.

- **Using irrelevant keywords and terms in Meta Tags**

Using completely unrelated keywords in metatags was a preferred method to raise a website's ranking because the worth of websites was determined by how many "eyeballs" saw the site. Even if a website was entirely about houseplants, it could still rank highly if the webmaster used popular terms like "sex" or "MP3" in his or her tags. It goes without saying that this prompted the search engines to once again adjust, and they started punishing websites that employed keyphrases that did not appear or were unrelated to the text in the website.

- **Tiny and invisible text**

This was a well-liked tactic. It might be very challenging to utilise every keyword and keyphrase where you want in your text because your web pages need to be appealing to visitors as well as search engines. As a result, some website owners employed "invisible text"-text that was written in the background colour of the web pageto add content to their online pages. The engines could consequently see it, but the human eye could not. A related tactic was "tiny text," which was text that was so small that it appeared on the screen as a line of dots or periods but was still recognised as text by search engines. This method is prohibited by the search engines.

- **Meta refresh tag**

Cloaking on a low-tech level is possible with the meta refresh tag. It was used to conceal to users' "doorways" or "jump pages." Within a

predetermined amount of time, typically less than one second, the tag itself automatically switched users to another page. Website owners were able to conceal their doorway pages from consumers by keeping the time increment very short (2 milliseconds, for example). The search engines would spider the doorwaypage, and then follow the link while visitors got another, often completely different page. The adult business found this to be highly popular.You may type in "Finance Advisor," get a list of results that are pertinent to your search, pick one, and then be taken to a pornographic website. As there were so many different redirect configurations, the administrators of the majority of search engines decided to outlaw all redirect pages. It is expected that users would view the same sites that search engines do.

- **Excessive search engine submission**

Most search engines have a cap on how many entries from a specific website they will accept in a given time frame. Occasionally, it's for several contributions made within a single day, week, or month. When a search engine receives an excessive number of submissions for the same Site within a given time frame, this is known as excessive submissions. When search engines still favoured websites that were "active," re-submission would have your site re-indexed. When you resubmitted, your website was considered "active." Today, you risk having your website blacklisted if you go over their limits. Because many website owners link their sites to numerous free submission platforms, it is highly vital to prevent this.

Oversubmitting a website could have the exact reverse of what is intended and should be avoided. An essential component of search engine optimization is submission. Don't attempt to overdo it; just do it right. You can overdo it in many other SEO areas and make the process more effective. You can put more effort into link development and spend more time producing content-rich text for the website. But if you submit too many things, your hard work will be undone.

3.6 SEARCH ENGINE OPTIMIZATION

The art and science of search engine optimization (SEO) involves raising a page's position in search engines like Google. Ranking higher in search engines can enhance a website's traffic because search is one of the primary ways users find material online.

Paid advertisements are frequently displayed at the top of the results page in Google and other search engines, followed by the regular results, or what search marketers refer to as the "organic search results." To distinguish it from traffic that comes from paid search, SEO traffic is frequently referred to as "organic search traffic." Search engine marketing (SEM), often known as pay-per-click, is another name for paid search (PPC).

3.6.1 Benefits of SEO

Online marketing relies heavily on search engine optimization because it is one of the main ways that people access the internet.

A site will often see greater traffic if it can move up the list of search results, which is given in alphabetical order. For instance, the top result for a normal search query will receive between 40 and 60 percent of all traffic, whereas results two and three will receive far less traffic overall. Just 2 to 3 percent of searchers click past the first page of results. A website may thus see an increase in visitors and perhaps prospective business as a result of even a slight improvement in search engine results.

As a result, a lot of companies and website owners try to manipulate the search results to have their website appear above those of their rivals on the search results page (SERP). Here's where SEO comes into play.

3.6.2 Working of SEO

For each query, search engines like Google utilise an algorithm or set of rules to decide which pages to display. To decide the rankings of their SERPs, these algorithms, which have become incredibly complicated over time, consider hundreds or even thousands of different ranking indicators. Search engines, however, base their evaluation of a site's quality and where it should rank on three key metrics:

- **Links:** The ranking of a website in Google and other search engines is significantly influenced by links from other websites. Due to the fact that website owners are unlikely to link to low-quality websites, a link can be viewed as a vote of approval from other websites. In the view of search engines, websites that receive links from numerous other websites gain authority (referred to as "PageRank" by Google), particularly if the websites linked to them are also authoritative.

- **Content:** Search engines examine a webpage's content in addition to its links to evaluate whether it is appropriate for a given search query. Making content that is focused on the keywords that users of search engines are looking for is a big aspect of SEO.

- **Page structure:** Page structure makes up the third essential element of SEO. Because HTML is the language used to create webpages, the way the HTML code is organised can affect how well a search engine understands a page. Site owners can take steps to enhance the SEO of their website by include pertinent keywords in the page's title, URL, and headers and by ensuring that a site is crawlable.

In order to rank higher in the search results, the search engine optimization method entails tweaking each of these fundamental elements of search engine algorithms.

3.6.3 SEO techniques

The first step in raising a site's search ranks is understanding how search engines operate. Using different SEO strategies to enhance the site for search is necessary to actually raise a site's ranking:

- **Keyword research -** Search engine optimization (SEO) frequently begins with keyword research, which entails determining which keywords a site already ranks for, which phrases its rivals rank for, and which additional terms potential buyers are using. Finding the keywords that users enter into Google and other search engines can give guidance on what existing material should be optimised and what new content should be produced.

- **Content marketing -** This strategy is used after potential keywords have been found. This could involve developing new content or upgrading already existing content. Because high-quality content is valued by Google and other search engines, it's crucial to analyse existing content, produce engaging content that offers a satisfying user experience, and increase your chances of appearing higher in search engine results. A good piece of content is more likely to be linked to and shared on social media.

- **Link building -** Getting high-quality backlinks is one of the fundamental SEO strategies since links from other websites, or "backlinks" in SEO lingo, are one of the main ranking factors in Google and other major search engines. In order to do this, it may be necessary to promote quality content, connect with other websites and cultivate relationships with webmasters, submit websites to pertinent online directories, and secure press coverage to entice connections from other websites.

- **On-page optimization -** In addition to off-page elements like links, enhancing the structure of the page itself can have a significant positive impact on SEO and is a factor that is completely under the webmaster's control. Typical on-page optimization strategies include altering the website's title tag to contain pertinent search terms, optimising the URL of the page to include keywords, and describing images using the alt attribute. Meta tags, like the meta description tag, can be updated to improve a page's click-through rate from the SERPs even if they don't directly affect search rankings.

- **Site architecture optimization -** Internal links, or links within one's own website, are just as important for SEO as external links are. In order to increase a page's relevance for particular phrases, a search engine optimizer can enhance a site's SEO by ensuring that crucial sites are connected to and that appropriate anchor text is used in those connections. A great technique for larger sites to aid search engines in finding and crawling all of the site's pages is by creating an XML sitemap.

- **Semantic markup -** Optimizing a website's semantic markup is another SEO tactic that SEO specialists use. To explain the meaning behind the material on a page, such as identifying the author of a piece of content or the topic and type of content on a page, semantic markup (such as Schema.org) is employed. Semantic markup can assist in obtaining rich snippets, such as additional text, review ratings, and even photos, shown on the search results page. Rich snippets in the SERPs don't affect search ranks, but they can increase search CTR, which raises organic traffic.

3.6.4 SEO tools

As a somewhat technical profession, SEO relies on a variety of programmes and tools to assist with website optimization. Here are a few frequently used tools, both free and paid:

- **Google Search Console-** Itis a free tool offered by Google that is a staple in the SEO's toolbox. Google Search Console was formerly known as "Google Webmaster Tools." GSC may assist in locating and resolving technical issues on the website and gives rankings and traffic reports for top keywords and pages.

- **Google Ads Keyword Planner -** Another free tool offered by Google as a component of its Google AdWords offering is the Keyword Planner for Google Ads. While being made for paid search, it may be an excellent SEO tool because it offers keyword ideas and keyword search volume, both of which are useful for conducting keyword research.

- **Backlink analysis tools -**There are many backlink analysing tools available, but the two most popular ones are AHREFs and Majestic. When link building, users can utilise backlink analysis tools to discover fresh links by looking at which websites link to their own or those of competitors.

- **Platforms for SEO -** There are many different platforms for SEO that combine many of the technologies required for SEO to optimise websites. Siteimprove, Moz, BrightEdge, Searchmetrics, and Linkdex are a few of the most well-known. These tools assist with keyword research, tracking keyword rankings, finding on-page and off-page SEO opportunities, and a variety of other SEO-related duties.

- **Social media -** Although the majority of social media platforms don't directly affect SEO, they can be a useful tool for networking with other webmasters and developing connections that may result in chances for link building and guest posting.

3.7 WEB SIZE MEASUREMENT

When defining the size of your Web page and the size of the various elements (most frequently fonts, but there are other elements as well), you

utilise three units of size (essentially width and length): pixels (px),ems (em), percentages (%).

- **Pixel**

A computer screen's display is made up of small dots called pixels. On a computer screen, everything that appears is represented by pixels. It's near to an absolute size because it varies little from computer to computer. 16px is a standard font size. Pixels have some disadvantages: People with visual issues may become frustrated by the inability of Internet Explorer users to resize text that has been set in pixels, and sometimes printing a Web page that has been set in pixels can also lead to issues. Users of IE7 and IE8 as well as Opera can resize their full page using "page zoom." Although it doesn't completely address the issue, it does help.)

- **Em**

As ems have a relative size, their size varies according to the size of the elements to which they are "related." Mostly speaking, it has to do with font size. The size 2em is therefore twice as large as the current font size.

Setting the base font size for your page is generally a smart idea. Although users occasionally reset their browser's display size to suit their needs, most browsers have a base size of 16px by default. Just set the font size to 16px so that it would look well on as many computers as possible (or whatever suits you).The use of ems is regarded by many designers as excellent practise. The disadvantage is that occasionally Internet Explorer 6 makes fonts set in ems appear much larger (or smaller) than they actually are.

- **Percentages**

This unit of measurement is considerably more arbitrary. You might use a font that is 80% the size of the primary typeface. In other words, if your main font size is, let's say, 16px, text displayed as 80% would size in at roughly 13px. Percentages are determined relative to the inherited size of the parent text. If your primary font size is 16px, your inherited element is at 80%, and you have another element inherited from the second one at, say, 80% again, the cumulative font size will be around 10px.

Owen Briggs, a web designer, did some research and provided 264 screenshots as examples. He has also given us his suggestions, which are a combination of percentages and ems. Although dated 2003, it still functions.

We almost exclusively use pixels, ems, and percentages even though there are alternative absolute ways to measure the size of items, such as points, picas, centimetres, millimetres, and others.

3.8 SEARCH ENGINE ARCHITECTURES

Crawler-indexer architecture is centralised in most search engines. The majority of search engine implementations are not accessible to the general public.There are yet still some to be found. These are AltaVista, Harvest and Google.

3.8.1 Alta Vista Architecture

The AltaVista search engine is discussed in this section as an illustration of how this architecture functions. Running locally on a computer, the crawler's job is to make queries to distant Web servers. To respond to user inquiries, the index is utilised centrally. The software architecture of AltaVista is depicted in the following diagram 3. It is split into two sections. The query engine and user interface; and the crawler and the indexer.

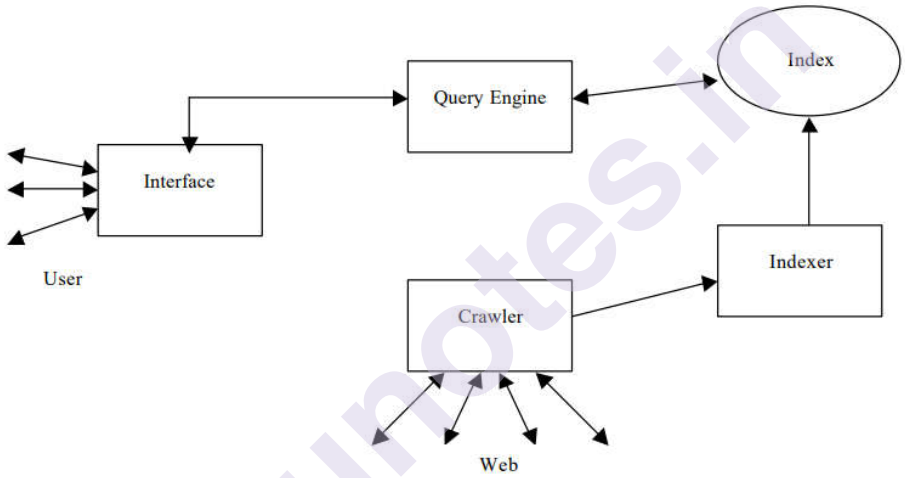

Figure 3:Alta Vistaarchitecture

Twenty processors were used by AltaVista in 1998. Around 500 GB of hard disc capacity and 130 GB of RAM are available on every processor. More than 75% of these resources are only used by the query engine.

This architecture suffers from two issues. Data collection in the dynamic Online context, which makes advantage of overloaded communication lines and high server load, is the first issue. The amount of data is the second issue. The crawler-indexer architecture cannot handle the expected future development of the Web.

3.8.2 HarvestArchitecture

The crawler-indexer architecture comes in a number of different forms. Harvest is the name of one of the variations. The most significant variation that employs distributed architecture to collect and disseminate data is called Harvest. The US National Academy of Sciences, NASA, the CIA, and the US Government Printing Office all utilise it. Moreover, the Harvest Cache from Network Appliances and the Netscape Catalog Server are also commercial versions of Harvest.

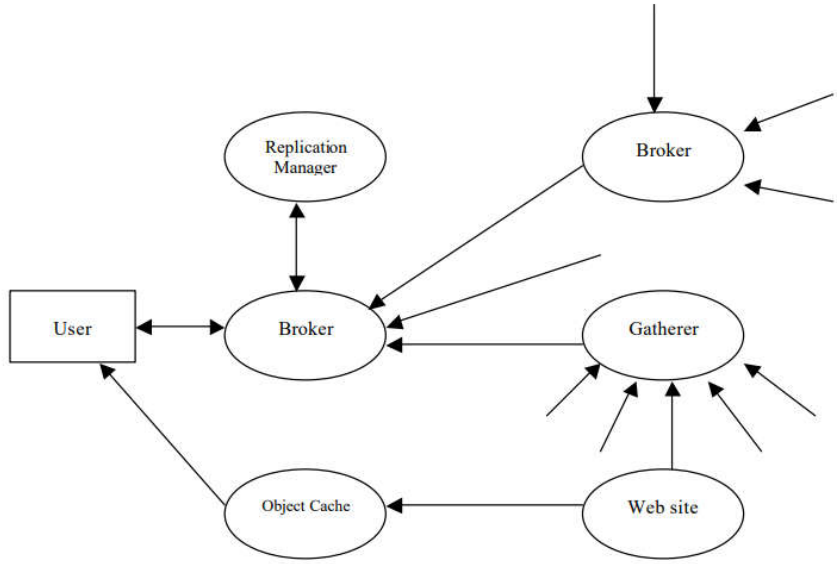

Figure 4: Harvest architecture

Harvest offers two key components: gatherers and brokers, as seen in Figure 4. Gatherers' duties include gathering and extracting indexing data from one or more Web servers. The Harvest system establishes gathering times. The name of the event, Harvest, denotes that the times are cyclical. Brokers' responsibilities include providing the indexing system and the query interface for the obtained data. To update their indices, brokers acquire data from gatherers or other brokers. Brokers can also filter information and send it to others, saving time for other brokers.

Network traffic and server workload can be balanced based on gatherer and broker settings. A lot of the vocabulary and scaling issues with generic indices are avoided by the harvest system, which creates topic-specific brokers and concentrates the index contents. The system also offers an object cache and a replication manager to replicate servers in order to increase user base scalability (to reduce network and server load).

3.8.3 GoogleArchitecture

The word googol, which signifies 10^{100}, is whence the name Google is derived. The hypertext framework is frequently used by the Google search engine (www.google.com). It asserts that its findings are superior to those of current search engines. There are over 24 million pages cited.

In order to maximise efficiency, Google is primarily written in C/C++. It can function on Linux or Solaris systems. Figure 5 depicts the architecture.

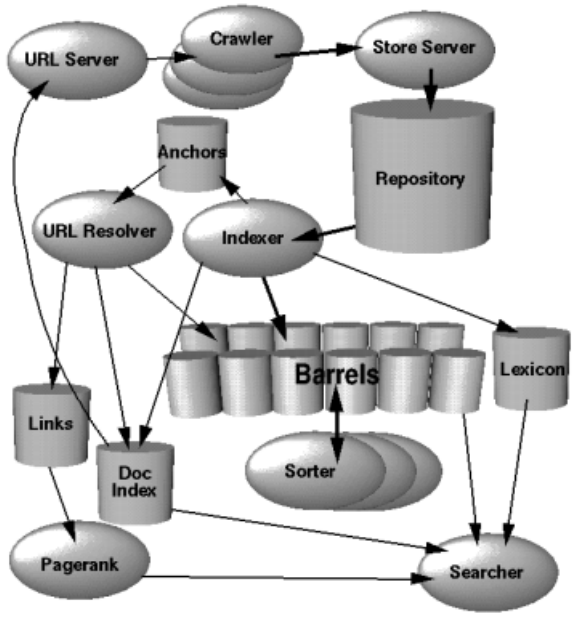

Figure 5: Google architecture

Lists of URLs are sent by the URL Server for the crawlers to retrieve. According to the list, the crawlers download pages, which they then deliver to the store server.The pages are compressed by the Store Server before being stored in the repository. Every time a new URL is extracted from a Web page, a new docID, also known as an associated ID number, is assigned. An indexing task is carried out by the index. It parses the pages afteruncompressing and reading the repository. Every page is divided into a collection of word occurrences known as hits. The hits provide details about a word, including its placement in the document, an estimated font size, and capitalization.The indexer divides up these hits into a number of "barrels" and produces a forward index that is only partially sorted (much like bucket sort). Every link on every Web page is extracted, and the anchors file contains all the pertinent data about them. The anchors file includes the content of each link as well as information about where it points from and to. The URL Resolver then reads the anchors file, changes the relative URLs to absolute URLs, and then converts those to docID. The forward index is updated to include the anchor text and docID. For the purpose of storing links and docIDs, it creates a links database. All of the documents' PageRanks are calculated using the database.

To create the inverted index, the Sorter takes the barrels and sorts them according to wordID rather than docID. A list of wordIDs and offsets into the inverted index is also produced by the Sorter.

This list and the lexicon created by the indexer are input into a software called DumpLexicon, which creates a new lexicon that the searcher can utilise. The searcher is managed by a Web server and employs the inverted index, PageRanks, and the lexicon created by DumpLexicon to respond to questions.

In 1998, Google had downloaded approximately 24 million Web pages. Its storage capacity was 108.7 GB with a repository and 55.2 GB without one. Google typically took between 1 and 10 seconds to respond to a query.

3.9 SUMMARY

The relevance of the results a search engine returns determines how valuable it is. Despite the fact that there may be millions of Web pages using a specific word or phrase, some may be more authoritative, popular, or relevant than others. The majority of search engines use techniques to rank the results such that the "better" results are displayed first.

There are significant differences amongst search engines in how they choose which pages are the best matches and what order the results should be displayed. As Internet usage changes and new ways are developed, the methods likewise change with time. The chapter also provides reasons why we need to study search engines, and it provides relevant references for readers to proceed further. More important, the readers should try out different search engines that are available today

3.10 LIST OF REFERENCES

1] Introduction to Information Retrieval, C. Manning, P. Raghavan, and H. Schutze, Cambridge University Press, 2008

2] Modern Information Retrieval: The Concepts and Technology behind Search, Ricardo Baeza -Yates and Berthier Ribeiro – Neto, 2nd Edition, ACM Press Books 2011.

3] Search Engines: Information Retrieval in Practice, Bruce Croft, Donald Metzler and Trevor Strohman, 1st Edition, Pearson, 2009.

4] Information Retrieval Implementing and Evaluating Search Engines, Stefan Buttcher, Charles L. A. Clarke and Gordon V. Cormack, The MIT Press; Reprint edition (February 12, 2016).

3.11 UNIT END EXERCISES

1] Discuss on the overview of web search.
2] Explain theweb engine structure/ architecture.
3] Write a note on user interfaces.
4] Explain the concept of Paid placement.
5] What do you mean by Search engine spam?
6] Write a detailed note on Search engine optimization.
7] Explain the working of SEO along with its benefits.
8] Discuss different tools and techniques associated with SEO.
9] Explain the concept ofWeb size measurement.
10] Write a note onAltaVista Architecture
11] Explain the Harvest Architecture.
12] Write a note on Google Architecture.

❖❖❖❖

4

XML RETRIEVAL

Unit Structure

4.0 OBJECTIVES

- To understand the basic XML concepts
- To get familiar with the challenges along with the model and its evaluation in XML retrieval process

4.1 INTRODUCTION

Relational databases and information retrieval systems are frequently compared.In the past, IR systems have traditionally retrieved data from unstructured text, which we define as "raw" text without markup. Relational data, or sets of records with values for predefined criteria like employee number, title, and income, is what databases are made for. Information retrieval and database systems have fundamentally different retrieval models, data structures, and query languages, as demonstrated in Table 1.

	RDB search	unstructured retrieval	structured retrieval
objects	records	unstructured documents	trees with text at leaves
model	relational model	vector space & others	?
main data structure	table	inverted index	?
queries	SQL	free text queries	?

► **Table 1** RDB (relational database) search, unstructured information retrieval and structured information retrieval. There is no consensus yet as to which methods work best for structured retrieval although many researchers believe that XQuery will become the standard for structured queries.

A relational database is best able to handle some highly structured text search situations, such as when you wish to identify all employees who are involved in billing and the employee table has an attribute for short textual job descriptions. The SQL query in this instance is:

```
select lastname from employees where job_desc like 'invoic%';
```

could be more than enough to meet your information needs with excellent precision and recall.

Therefore, it is preferable to model many text-based structured data sources as structured documents as opposed to relational data. Searching via these organised documents is referred to as structured retrieval. In structured retrieval, queries can be either structured or unstructured; nevertheless, for the sake of this chapter, we'll assume that the collection only contains structured documents. Examples of structured retrieval include output from office suites like OpenOffice that save documents as marked up text, patent databases, blogs, and text in which entities like people and locations have been tagged (in a technique called named entity tagging).

Extensible Markup Language, or XML, which is now the most popular such standard, will be the only one we examine for encoding structured texts. The nuances that set XML apart from other markup languages like HTML and SGML will not be discussed. The majority of our discussion in this chapter, however, applies to markup languages in general.

4.2 BASIC XML CONCEPTS

An XML file is a labelled, ordered tree. With an opening and closing tag, the tree's nodes are each written as XML elements. There may be one or more XML attributes for an element. The two tags <scene ...> and </scene> contain the scene element in the XML document shown in Figure 1. It has two child elements, title and poem, as well as an attribute number with the value vii.

```
<play>
<author>Shakespeare</author>
<title>Macbeth</title>
<act number="I">
<scene number="vii">
<title>Macbeth's castle</title>
<verse>Will I with wine and wassail ...</verse>
</scene>
</act>
</play>
```

Figure 1: An XML document

Figure 2 shows Figure 1 as a tree. Shakespeare, Macbeth, and Macbeth's castle are only a few examples of the text that makes up the tree's leaf nodes. The interior nodes of the tree represent either the metadata functions or the document's structure (title, act, and scene) (author).

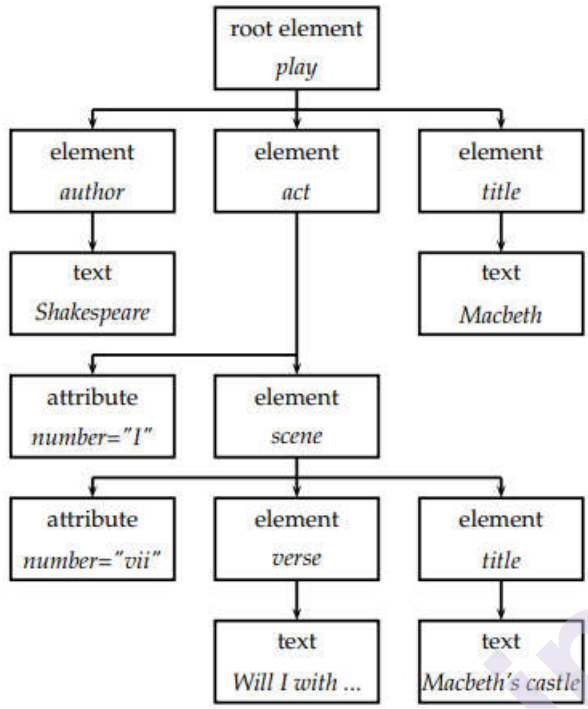

Figure 2: The XML document in Figure 1 as a simplified DOM object

The XML Document Object Model, or DOM, is the industry standard for XML document access and processing. Elements, attributes, and text contained within elements are represented by the DOM as nodes in a tree. The XML document in Figure 1 is depicted in a more condensed DOM format in Figure 2. An XML document can be processed using a DOM API by beginning at the root element and working our way down the tree from parents to children.

A standard for listing pathways in an XML document collection is called XPath.In this chapter, pathways will also be referred to as XML contexts or just contexts. For our purposes, only a small portion of XPath is required. All nodes with that name are selected by the XPath expression node. Act/scene picks all scene components whose parent is an act element since the elements of a path are separated by slashes. A path can be interrupted by any number of components, as seen by the double slashes play//scene, which selects all scene elements that appear in a play element. This set in Figure 2 only has one scene element, which is accessed from the top using the path play, act, scene.

The idea of a schema is also important in this chapter. The permitted structure of XML documents for a certain application is constrained by a schema. Shakespeare's plays might contain a schema that states that only acts and scenes have the number attribute and that scenes can only be offspring of acts. XML DTD (document type definition) and XML Schema are two specifications for XML document schemas. Users can only create structured queries for an XML retrieval system if they are at least somewhat familiar with the collection's schema.

Narrowed Extended XPath I, sometimes known as NEXI, is a popular format for XML queries. Figure 3 provides an illustration. We display the query on four lines fortypographical convenience, but it is intended to be read as one unit withoutline breaks. In particular, //section is embedded under //article.

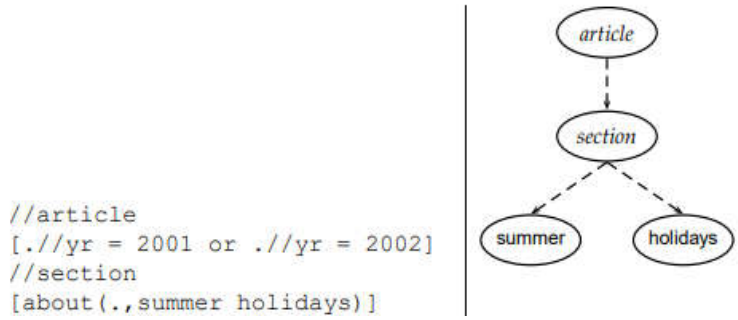

Figure 3: An XML query in NEXI format and its partial representation as a tree

The search criteria in Figure 3 call for finding summer vacation-related sections of articles from 2001 or 2002. Double slashes, like in XPath, denote that any number of elements may obstruct a route. The element that a clause alters is indicated by the dot enclosed in square brackets.The clause [.//yr = 2001 or .//yr = 2002] modifies //article. Thus, the dot refers to //article in this case. Similarly, the dot in [about(., summer holidays)] refers to the section that the clause modifies.

Relational attribute constraints govern the two year conditions. Only articles with the year attribute 2001 or 2002 (or elements with the year attribute 2001 or 2002) should be taken into consideration. The about clause serves as a ranking restriction, requiring that sections that appear in the appropriate kind of article be ranked according to how pertinent they are to the theme of summer vacations.

Prefiltering or postfiltering is how we typically handle relational attribute restrictions: We simply remove all elements from the result set that do not adhere to the requirements. Instead of discussing how to accomplish this effectively in this chapter, we will concentrate on the core information retrieval issue in XML retrieval, which is how to rank documents in accordance with the relevant criteria stated in the about conditions of the NEXI query.

Relational attributes can be disregarded if we want to represent documents as trees with only element nodes as nodes. In other words, all attribute nodes, such as the number attribute in Figure 1, are removed from the XML document. A subtree of the document in Figure 1 is displayed as an element-node tree in Figure 4(labeled d1).

In a similar manner, we can express questions as trees. Because users pose inquiries by building objects that satisfy the same formal description as documents, this query-by-example method of query language design was developed. Finding books with titles that rank highly for the keywords

Julius Caesar is shown in Figure 4's q1 search. Finding books with author elements that rank highly for Julius Caesar and title components that rank highly for the Gallic War is the second question.

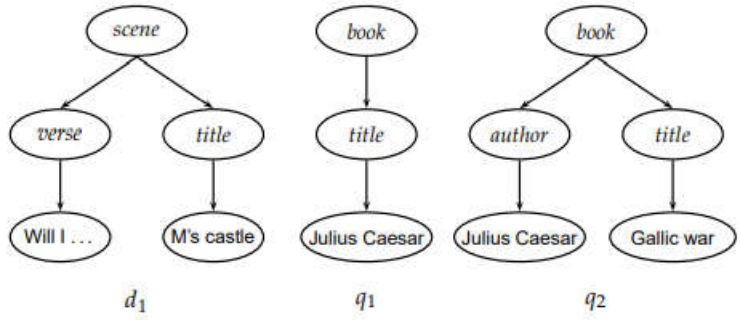

Figure 4: Tree representation of XML documents and queries

4.3 CHALLENGES IN XML RETRIEVAL

Users want us to return portions of documents (i.e., XML elements) rather than full documents, as IR systems often do in unstructured retrieval. This presents the first challenge in structured retrieval. Should the full play, the act, or the scene in Figure 2 be returned if we search Shakespeare's plays for Macbeth's castle? The user is most likely searching for the scene in this instance. On the other hand, a general search for Macbeth ought to turn up the play itself and not a component.

The organised document retrieval principle is one standard for determining which portion of a document is most appropriate: retrieval of structured documents as a principle. In order to best respond to a query, a system should always obtain the most specific portion of the document.

This idea drives a retrieval technique that only descends to the level of the smallest unit containing the information sought. Algorithmically using this principle, nevertheless, can be challenging. Have a look at Figure 2 with the title "Macbeth" applied.Because they both contain the word "Macbeth," the title of the tragedy, Macbeth, and the title of Act I, Scene vii, Macbeth's castle, are both successful hits. However, in this instance, the upper node, the tragedy's title, is favoured. Selecting the appropriate level of the tree to address a query might be challenging.

The question of which portions of a document to index is related to the one of which portions to return to the user. The appropriate document unit in unstructured retrieval is typically obvious: desktop files, emails, online web pages, etc. There are several alternative methods for defining the indexing unit in structured retrieval.

One strategy is to organise nodes into separate pseudodocuments, as seen in Figure 5. Books, chapters, and sections have been specified as indexing units in the example, but there is no overlap. For instance, only the portions of the tree dominated by books that are not already included in other indexing units are included in the leftmost dashed indexing unit. The drawback of this strategy is that because pseudodocuments are not

coherent pieces, the user may not understand them. For instance, the leftmost indexing unit in Figure 5 combines the class, author, and title elements, which were formerly three separate items.

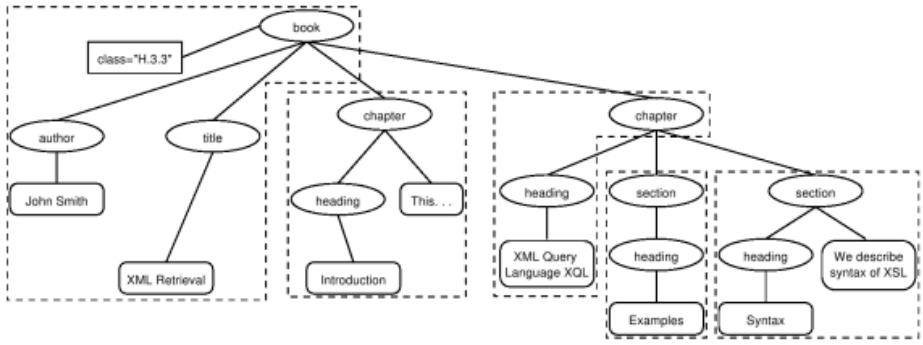

Figure 5: Partitioning an XML document into non-overlapping indexing units

One of the biggest parts, such as the book element in a collection of books or the play element for Shakespeare's works, can also serve as the indexing unit. The best-hitting subelement for each book or play can then be found using post-processing on the search results. For instance, the search for Macbeth's castle might lead to the play Macbeth, which we can then post-process to find the best-matching subelement in act I, scene vii. Due to the fact that the relevance of a book as a whole is frequently not a strong predictor of the importance of small subelements within it, this two-stage retrieval technique unfortunately fails to deliver the optimal subelement for many queries.

We can search all leaves, choose the most pertinent ones, and then expand them to larger units in post-processing instead of retrieving huge units and identifying subelements (top down) (bottom up). In order to determine whether to return the title, the scene, the act, or the play for the query Macbeth's castle in Figure 1, we would first obtain the title Macbeth's castle. The significance of a leaf element is frequently not a good predictor of the relevance of elements it is included in, which is a problem shared by this strategy and the previous one.

Indexing all items is the strategy with the fewest restrictions. This is a problem as well. Many XML elements, such as typographical elements like absolutely or an ISBN number that cannot be understood without context, are not meaningful search results. Moreover, indexing every element will result in a lot of duplicated search results. We would return all of the play, act, scene, and title components on the path between the root node and Macbeth's castle for the query Macbeth's castle and the document in Figure 1.Once directly and three times as a component of other elements, the leaf node would then appear four times in the result set. Elements that are nestled inside of one another are referred to as nested.It is not very user-friendly to return redundant nested components in a list of returned hits.

It is usual to limit the set of elements that are eligible to be returned due to the redundancy generated by nested elements. The following are some restrictions:

- Remove all minor components
- Remove all element kinds that users do not examine (to do this, an operating XML retrieval system that records this information is needed).
- Eliminate all element kinds that assessors typically deem irrelevant (if relevance assessments are available)
- only maintain the element kinds that a system designer or librarian has determined to be helpful search results.

The majority of these methods still provide result sets with nested components.As a result, in order to eliminate repetition, we could want to remove some pieces in a postprocessing phase. As an alternative, we can collapse a number of nested elements in the results list and highlight the search phrases to highlight the pertinent portions. A medium-sized element (such as a section) is scanned slightly more slowly when query terms are highlighted than a small subelement (e.g., a paragraph). Hence, it is sufficient to display the section if both the section and the paragraph appear in the results list. This strategy also has the benefit of presenting the paragraph along with its context (i.e., the embedding section).Even though the paragraph answers the question on its own, the context may be useful in understanding the paragraph (e.g., the source of the information reported).

Redundancy is less of an issue if the user is aware of the collection's structure and is able to identify the kind of element they want, as few nested elements have the same type. However, as we noted in the introduction, users frequently lack basic knowledge of how to construct structured queries or are unaware of the name of an element in the collection (e.g., Is the Vatican a country or a city?).

We may need to discriminate between multiple contexts of a term when computing term statistics for ranking, in particular inverse document frequency (idf) statistics, which presents a barrier in XML retrieval related to nesting.When used to refer to the plural of gate, the phrase Gates under the node author has nothing to do with an occurrence under a content node like section. In this case, computing Gates' single document frequency doesn't make much sense.

The computation of idf for XML-context/term pairings, such as author#"Gates" and section#"Gates," is one approach. Unfortunately, this method will encounter issues with sparse data, as many XML-context pairings only sometimes occur, making it difficult to accurately estimate df.To distinguish between contexts, it is a compromise to only take into account the term's parent node x and ignore the rest of the path from the root to x. This approach still contains context conflations that are dangerous. For instance, if both names share the parent node name, we do

not distinguish between names of authors and names of organisations. However, the majority of significant distinctions—for instance, author#"Gates" and section#"Gates"will be observed.

Since XML documents in IR applications sometimes originate from multiple sources, a collection of XML documents frequently contains multiple distinct XML schemas. Schema heterogeneity or diversity refers to this phenomenon, which creates even another difficulty. Same items may have distinct names, as seen in Figure 6, for example, creator in d2 vs. author in d3. Some instances may involve a different structural arrangement of the schemas: Firstname and lastname are intermediate nodes in d3, whereas author names are direct descendants of the node author in q3. Even though both documents are pertinent, q3 won't obtain either d2 or d3 if we use stringent matching of trees.Here, any kind of approximate element name matching along with semi-automatic document structure matching can be helpful. Human editing of element correspondences across several schemas typically performs better than computerised techniques.

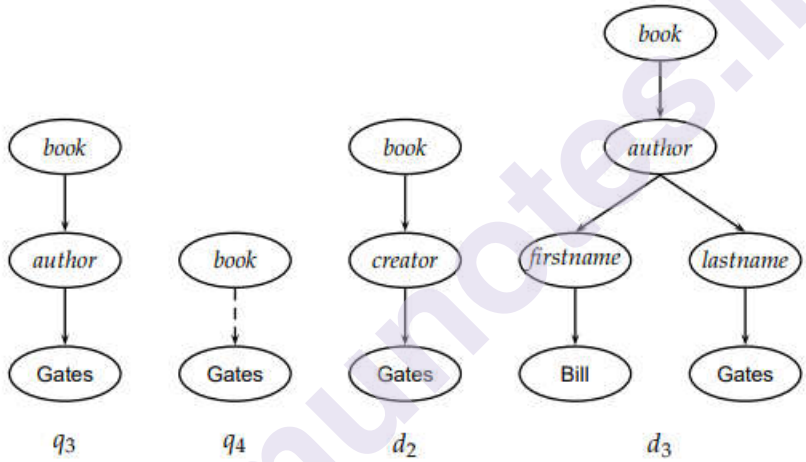

Figure 6: Schema heterogeneity: intervening nodes and mismatched names

One cause of query-document mismatches like q3/d2 and q3/d3 is schema heterogeneity. Another factor is that, as previously indicated, users frequently lack familiarity with the element names and structures of the schemas of the collections they search. This presents a problem for XML retrieval interface design.

The user interface should ideally display the collection's tree structure and enable users to specify the elements they are querying. If we adopt this strategy, creating the query interface for structured retrieval will be more difficult than creating a search box for keyword queries for unstructured retrieval.

4.4 A VECTOR SPACE MODEL FOR XML RETRIEVAL

We offer a straightforward vector space model for XML retrieval in this section.In Figure 4, we want a book titled Julius Caesar to be a match for

q1 and no match (or a lower weighted match) for q2, in order to account for structure in retrieval. Caesar's vector space would only have one dimension in unstructured retrieval. The title word Caesar and the author name Caesar must be separated for XML retrieval. One approach of accomplishing this is by encoding a word and its location inside the XML tree in each dimension of the vector space.

This illustration is shown in Figure 7. In order to divide each text node—which in our system is always a leafinto numerous nodes, one for each word, we first take each text node. As a result, Bill and Gates emerge from the leaf node Bill.The dimensions of the vector space for the lexicalized subtrees of documents, or subtrees with at least one vocabulary term, are then defined. The picture only depicts a portion of the possible lexicalized subtrees; there are others as well, such as the subtree matching to the entire document without the leaf node Gates. In this space of lexicalized subtrees, we can now describe queries and documents as vectors and compute matches between them.As a result, we may get XML using the vector space formalism. The primary distinction is that the dimensions of vector space in XML retrieval are lexicalized subtrees, whereas they are vocabulary terms in unstructured retrieval.

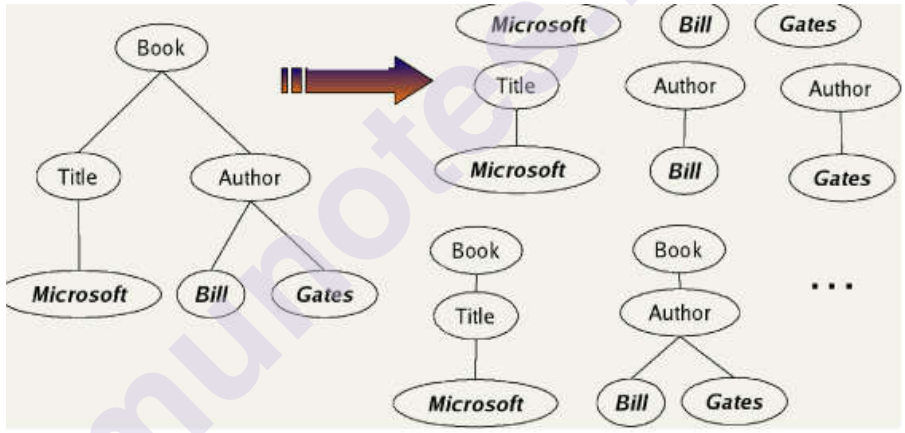

Figure 7: A mapping of an XML document (left) to a set of lexicalized subtrees (right)

The accuracy of query results trades off against the space's dimensionality. A conventional vector space retrieval system will receive many documents that do not match the structure of the query (for example, Gates in the title as opposed to the author element) if we trivially restrict dimensions to vocabulary items. The dimensionality of the space increases if we add a distinct dimension for each lexicalized subtree found in the collection. Indexing all paths that lead to a single vocabulary term, or all XML-context/term pairings, serves as a compromise.Such an XML-context/term pair is what we refer to as a structural term, and it is represented by the symbols <c,t>: an XML-context c and a vocabulary term t. Nine structural terms are present in the document in Figure 7. Seven are displayed (such as "Bill" and Author#"Bill"), whereas /Book/Author#"Bill" and /Book/Author#"Gates" are not. Bill and Gates' names are the leaves of a

lexicalized subtree, not a structural term. For structural terms, we employ the previously described pseudo-XPath notation.

We ensure that retrievalresults respect this preference by computing a weight for each match. A simple measure of the similarity of a path c_q in a query and a path c_d in a document is the followingcontext resemblance function C_R:

$$C_R(c_q, c_d) = \begin{cases} \frac{1+|c_q|}{1+|c_d|} & \text{if } c_q \text{ matches } c_d \\ 0 & \text{if } c_q \text{ does not match } c_d \end{cases} \quad (1)$$

where $|c_q|$ and $|c_d|$ are the number of nodes in the query path and document path, respectively, and c_q matches c_d iff we can transform c_q into c_d by inserting additional nodes. Two examples from Figure 10.6 are $C_R(c_{q_4}, c_{d_2}) = 3/4 = 0.75$ and $C_R(c_{q_4}, c_{d_3}) = 3/5 = 0.6$ where c_{q_4}, c_{d_2} and c_{d_3} are the relevant paths from top to leaf node in q_4, d_2 and d_3, respectively. The value of $C_R(c_q, c_d)$ is 1.0 if q and d are identical.

The final score for a document is computed as a variant of the cosine measure which we call SIMNOMERGE for reasons that will become clear shortly. SIMNOMERGE is defined as follows:

$$\text{SIMNOMERGE}(q,d) = \sum_{c_k \in B} \sum_{c_l \in B} C_R(c_k, c_l) \sum_{t \in V} \text{weight}(q,t,c_k) \frac{\text{weight}(d,t,c_l)}{\sqrt{\sum_{c \in B, t \in V} \text{weight}^2(d,t,c)}} \quad (2)$$

where V is the vocabulary of non-structural terms; B is the set of all XML contexts; and weight(q, t, c) and weight(d, t, c) are the weights of term t in XML context c in query q and document d, respectively. We compute the weights using one of the weightings such as $idf_t \cdot wf_{t,d}$. The inverse document frequency idf_t depends on which elements we use to compute df_t

The algorithm for computing SIMNOMERGE for all documents in the collection is shown in Figure 8.

```
SCOREDOCUMENTSWITHSIMNOMERGE(q, B, V, N, normalizer)
 1  for n ← 1 to N
 2  do score[n] ← 0
 3  for each ⟨c_q, t⟩ ∈ q
 4  do w_q ← WEIGHT(q, t, c_q)
 5     for each c ∈ B
 6     do if C_R(c_q, c) > 0
 7           then postings ← GETPOSTINGS(⟨c, t⟩)
 8                for each posting ∈ postings
 9                do x ← C_R(c_q, c) * w_q * weight(posting)
10                   score[docID(posting)] += x
11  for n ← 1 to N
12  do score[n] ← score[n] / normalizer[n]
13  return score
```

Figure 8: The algorithm for scoring documents with SIMNOMERGE

4.5 EVALUATION OF XML RETRIEVAL

The premier venue for research on XML retrieval is the INEX (INitiative for the Evaluation of XML retrieval) program, a collaborative effort that has produced reference collections, sets of queries, and relevance judgments. A yearly INEX meeting is held to present and discuss research results. TheINEX 2002 collection consisted of about 12,000 articles from IEEE journals. We give collection statistics in Table 2 and show part of the schema of the collection in Figure 9.The IEEE journal collection was expanded in 2005. Since 2006 INEX uses the much larger English Wikipedia as a test collection. The relevance of documents is judged by human assessors using the methodology appropriately modified for structured documents.

Table 2: INEX 2002 collection statistics

12,107	number of documents
494 MB	size
1995–2002	time of publication of articles
1,532	average number of XML nodes per document
6.9	average depth of a node
30	number of CAS topics
30	number of CO topics

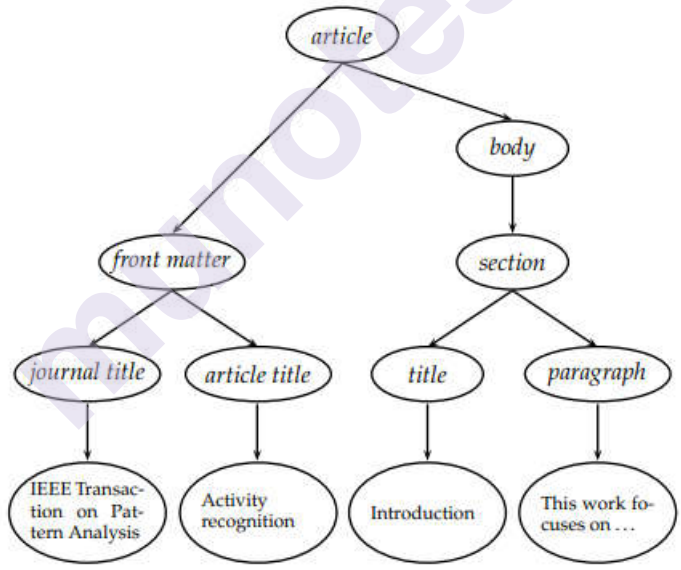

Figure 9: Simplified schema of the documents in the INEX collection

Two categories of information needs or topics in INEX are content-only or CO topics and content-and-structure (CAS) topics. CO topics are ordinary keyword searches as in unstructured information retrieval. CAS subjects include structural limitations in addition to keywords. We already see an example of a CAS topic in Figure 3. The keywords in this case are summer and holidays and the structural constraints indicate that the keywords exist in a section that in turn is part of an article and that this article has an embedded year property with value 2001 or 2002.

As CAS searches involve both structural and content criteria, relevance judgements are more complicated than in unstructured retrieval. INEX 2002 defined component coverage and subject relevance as orthogonal measures of relevance. The component coverage dimension checks if the element retrieved is "structurally" correct, i.e., neither too low nor too high in the tree. We distinguish four cases:

- Exact coverage (E): The information sought is the main topic of the component and the component is a meaningful unit of information
- Too small (S): The information sought is the main topic of the component, but the component is not a meaningful (self-contained) unit of information
- Too large (L): The information sought is present in the component, but is not the main topic
- No coverage (N): The information sought is not a topic of the component

The topical relevance dimension also has four levels: highly relevant (3), fairly relevant (2), marginally relevant (1) and nonrelevant (0). Components are judged on both dimensions and the judgments are then combined into a digit-letter code. 2S is a fairly relevant component that is too small and 3E is a highly relevant component that has exact coverage. In theory, there are 16 combinations of coverage and relevance, but many cannot occur. For example, a nonrelevant component cannot have exact coverage, so the combination 3N is not possible. The relevance-coverage combinations are quantized as follows:

$$\mathbf{Q}(rel, cov) = \begin{cases} 1.00 & \text{if} \quad (rel, cov) = 3\text{E} \\ 0.75 & \text{if} \quad (rel, cov) \in \{2\text{E}, 3\text{L}\} \\ 0.50 & \text{if} \quad (rel, cov) \in \{1\text{E}, 2\text{L}, 2\text{S}\} \\ 0.25 & \text{if} \quad (rel, cov) \in \{1\text{S}, 1\text{L}\} \\ 0.00 & \text{if} \quad (rel, cov) = 0\text{N} \end{cases}$$

The fact that binary relevance judgements, which are common in unstructured information retrieval, are inappropriate for XML retrieval is taken into account by this evaluation scheme. Although a 2S component only gives partial information and may be difficult to understand without further context, it partially satisfies the research question. The quantization function Q allows us to rank a component as partially relevant rather than forcing us to make a binary decision between relevant and irrelevant.

The number of relevant components in a retrieved set A of components can then be computed as:

$$\#(\text{relevant items retrieved}) = \sum_{c \in A} \mathbf{Q}(rel(c), cov(c))$$

Because we sum graded rather than binary relevance ratings, the usual definitions of precision, recall, and F from can be roughly applied to our modified meaning of relevant items retrieved.

4.6 TEXT-CENTRIC VERSUS DATA-CENTRIC XML RETRIEVAL

To match the text of the query with the text of the XML documents, the form of structured retrieval we examine in this chapter uses the XML structure as a framework. This is an example of a system designed with text-centric XML in mind. Although both text and structure are crucial, we give text a higher emphasis. By modifying unstructured retrieval techniques to handle extra structural limitations, we achieve this. Our strategy is predicated on the notion that XML document retrieval is characterised by I extensive text fields (for example, parts of a document), (ii) imperfect matching, and (iii) relevance-ranked returns. This use case does not work well with relational databases.

XML that is focused on data, on the other hand, primarily encodes numerical and non-text attribute value data. Most of the time, we wish to apply exact match requirements while querying data-centric XML. As a result, the structural features of XML documents and queries are highlighted. Among them is:

Identify workers whose monthly pay is the same as it was a year ago.

There is no need to rank this query. Since it is entirely structural, the user's information needs can probably be satisfied by an exact match between the salaries in the two time periods.

For data that are essentially text documents marked up as XML to capture document structure, text-centric techniques are appropriate. Due to the fact that most text papers contain some kind of intriguing structure-paragraphs, sections, footnotes, etc. this is emerging as the de facto standard for publishing text databases. Examples include assembly manuals, journal papers, the collected works of Shakespeare, and newswire pieces.

For data collections with complex structures that primarily comprise non-text data, data-centric techniques are frequently used. With proteomic data in bioinformatics or the depiction of a city map that, along with street names and other textual descriptions, constitutes a navigational database, a text-centric retrieval engine will struggle.

Joins and ordering constraints are two additional sorts of queries that are challenging to handle in a text-centric structured retrieval architecture. A join is necessary to find employees whose salaries have remained the same. The following query imposes
an ordering constraint:

Get the chapter on algorithms that comes after the one on binomial heaps in the book

This search depends on the arrangement of XML elements, specifically the arrangement of chapter elements under the book node. For XML, there are sophisticated query languages that support joins, ordering constraints, and numerical attributes. The most well-known of them is XQuery, a

language that the W3C wants to standardise. It is intended to have a wide range of applications in every field where XML is utilised. It is difficult to create an XQuery-based ranked retrieval system with the performance traits that consumers have grown accustomed to in information retrieval due to its complexity. One of the most active areas of XML retrieval research right now is this one.

Relational databases, especially joins, are better suited to handle numerous structural constraints (but ordering is also difficult in a database framework – the tuples of a relation in the relational calculus are not ordered). Because of this, relational databases are the foundation of the majority of data-centric XML retrieval systems. Using a relational database for XML retrieval is appropriate if text fields are brief, accurate matching fits user needs, and retrieval results in the form of unordered sets are acceptable.

4.7 SUMMARY

The content-based retrieval of XML-structured documents is known as XML retrieval, or XML information retrieval (eXtensible Markup Language). As a result, it is utilised to determine the significance of XML documents.We may need to discriminate between multiple contexts of a term when computing term statistics for ranking, in particular inverse document frequency (idf) statistics, which presents a barrier in XML retrieval related to nesting.

Almost fifty organisations from around the world are participating in the international campaign known as the Initiative for the Evaluation of XML Retrieval (INEX). It offers a way to assess retrieval programmes that give users access to XML content. Documents today comprise a blend of textual, multimedia, and metadata information.

Information interchange between computer systems, including webpages, databases, and outside applications, is supported by XML. Because the recipient may use the predefined rules to interpret the data accurately and effectively, it is simple to send data as XML files across any network.The objective is to make it feasible for generic SGML to be provided, received, and processed on the Web in the same way that HTML is currently possible.

4.8 LIST OF REFERENCES

1] Introduction to Information Retrieval, C. Manning, P. Raghavan, and H. Schutze, Cambridge University Press, 2008

2] Modern Information Retrieval: The Concepts and Technology behind Search, Ricardo Baeza -Yates and Berthier Ribeiro – Neto, 2nd Edition, ACM Press Books 2011.

3] Search Engines: Information Retrieval in Practice, Bruce Croft, Donald Metzler and Trevor Strohman, 1st Edition, Pearson, 2009.

4] Information Retrieval Implementing and Evaluating Search Engines, Stefan Buttcher, Charles L. A. Clarke and Gordon V. Cormack, The MIT Press; Reprint edition (February 12, 2016).

4.9 UNIT END EXERCISES

1] Explain the basic XML concepts.

2] What are the different challenges in XML retrieval?

3] Describe a vector space model for XML retrieval.

4] Explain the evaluation of XML retrieval.

5] Explain the difference between Text-centric versus data-centric XML retrieval.