

CBCS SCHEME

USN

1	C	R	2	2	C	1	0	1	9
---	---	---	---	---	---	---	---	---	---

BAI515B

Fifth Semester B.E./B.Tech. Degree Examination, Dec.2024/Jan.2025

Information Retrieval

Time: 3 hrs.

Max. Marks: 100

*Note: 1. Answer any FIVE full questions, choosing ONE full question from each module.
2. M : Marks , L: Bloom's level , C: Course outcomes.*

Module – 1				M	L	C
Q.1	a.	Explain the high level view of the software architecture of an IR system with a neat labelled block diagram.	10	L2	CO2	
	b.	Explain the processes of retrieval and ranking of documents.	10	L2	CO3	
OR						
Q.2	a.	Explain the following terms: (i) Query specification (ii) Query reformulation	08	L3	CO3	
	b.	Explain the visualization in search interfaces.	12	L2	CO1	
Module – 2						
Q.3	a.	With a neat diagram, explain the taxonomy of IR models.	10	L3	CO3	
	b.	Explain the following with reference to the Classic Information Retrieval: (i) The Boolean Model (ii) Term Weighting	10	L3	CO2	
OR						
Q.4	a.	Explain the Fuzzy Information Retrieval approach in detail.	10	L2	CO2	
	b.	Explain the following models briefly: (i) Neural Network Model (ii) Latent Semantic Indexing Model	10	L3	CO3	
Module – 3						
Q.5	a.	Explain precision and recall for a given information request 'I'.	08	L2	CO1	
	b.	Explain explicit and implicit feedback information in detail.	12	L3	CO2	
OR						
Q.6	a.	Explain implicit feedback through Global Analysis.	10	L2	CO2	
	b.	Explain the following terms with reference to the text properties: (i) Information theory (ii) Modeling Natural Language (iii) Text Similarity	10	L3	CO3	
Module – 4						
Q.7	a.	Explain Full Inverted Indexes in detail with suitable example.	10	L2	CO2	
	b.	Explain the following: (i) Signature Files (ii) Tries and suffix trees	10	L3	CO3	
OR						

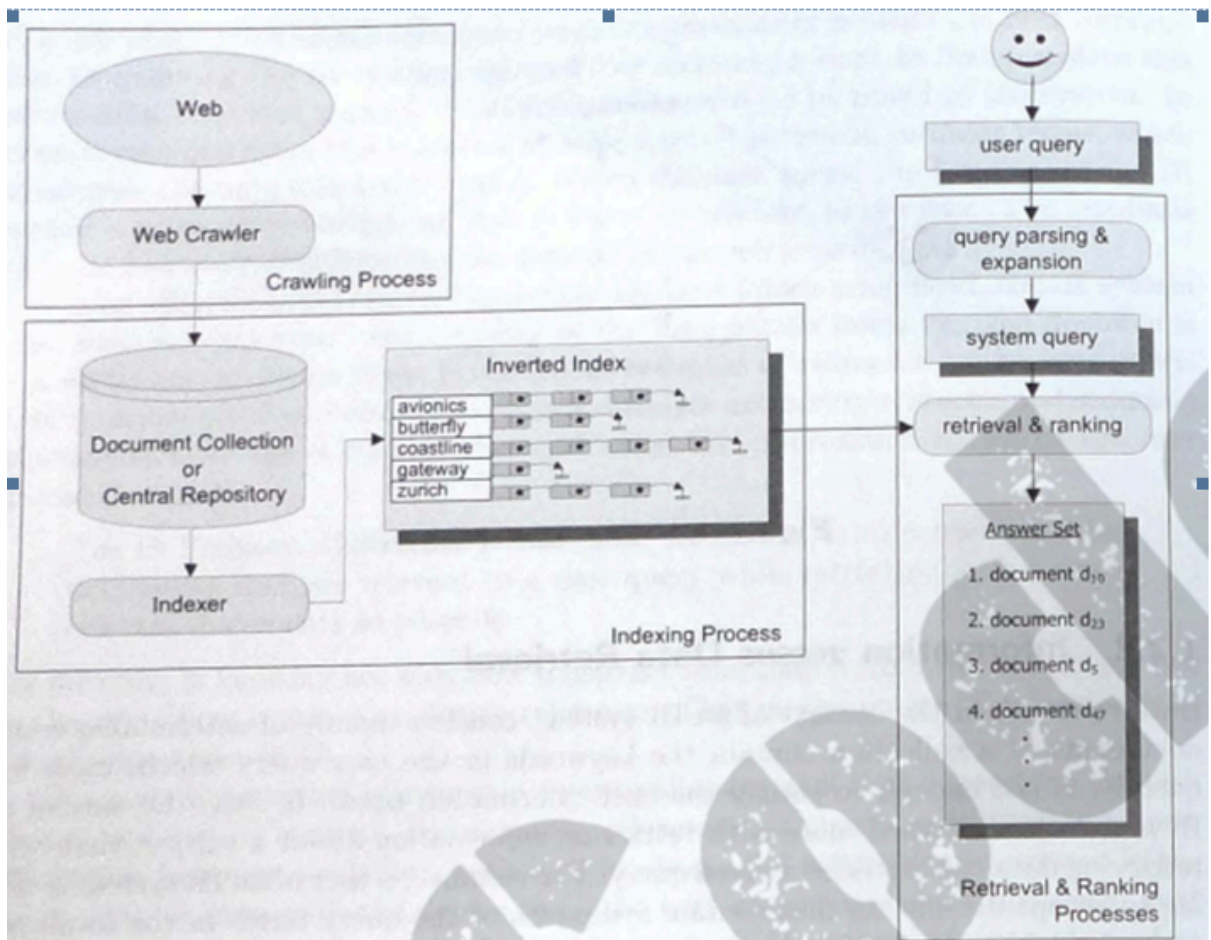
1 of 2

				BAI515B		
Q.8	a.	Explain suffix trees and suffix arrays.	08	L2	CO2	
	b.	Explain the following: (i) Faster Bit-Parallel Algorithms (ii) Multi-dimensional Indexing	12	L4	CO4	
Module – 5						
Q.9	a.	Explain Search Engine Architecture.	10	L3	CO2	
	b.	Explain the cluster based architecture for the search module with its key components briefly.	10	L3	CO3	
OR						
Q.10	a.	Explain the XML Retrieval Evaluation in detail.	12	L1	CO2	
	b.	Write notes on: (i) Structure of Web graph (ii) Link based ranking	08	L3	CO4	

Module 1

Question 1

a. Software Architecture of the IR System



-The first step is Assembling the document collection which may be private or crawled from **The Web** using a **Crawler Module**.

-The document collection is stored in the disk storage, usually referred to as the **central repository**.

-The documents in the central repository are then **indexed**, for fast retrieval and ranking. The most used index structure is an *inverted index* composed of all the **distinct words** of the collection and for each word, a list of documents that contain it.

-Once the document is **Indexed**, the **retrieval process** can be initiated. It consists of retrieving documents that satisfy a user query(searching) or a click in a hyperlink(browsing).

-To search a **user** first specifies a **query**, that reflects their information need.

-Next, the user **query is parsed**, and **expanded**. (with for instance spelling variants of a query word).

The expanded query is referred to as the **System Query**

-The system query is then processed against the index to **retrieve** a subset of documents.

-The Retrieved documents are then **ranked** and the top documents are returned to user.

>A systematic evaluation process allows fine tuning of the ranking algorithm, one evaluation is **comparing** the set of results produced by the **IR system** with results suggested by **human specialists**.

>To improve the ranking we might also **collect feedback** from users and use this information to change results.

b. The Retrieval and Ranking Process

Indexing Process

-Given the document representations, it is necessary to build an index of the text. Different structures might be used, but the most popular one is an inverted index.

-The steps required to generate the index compose of the indexing process

-The indexing process must be done "offline".

For which the Resources(time and storage) are amortized for every query that the retrieval system makes

Retrieval Process

-The user first specifies a query that reflects their information need.

-The query is then parsed and modified by operations that resemble those done to the documents.

-The expanded and modified query is then processed to obtain the set of retrieved documents.

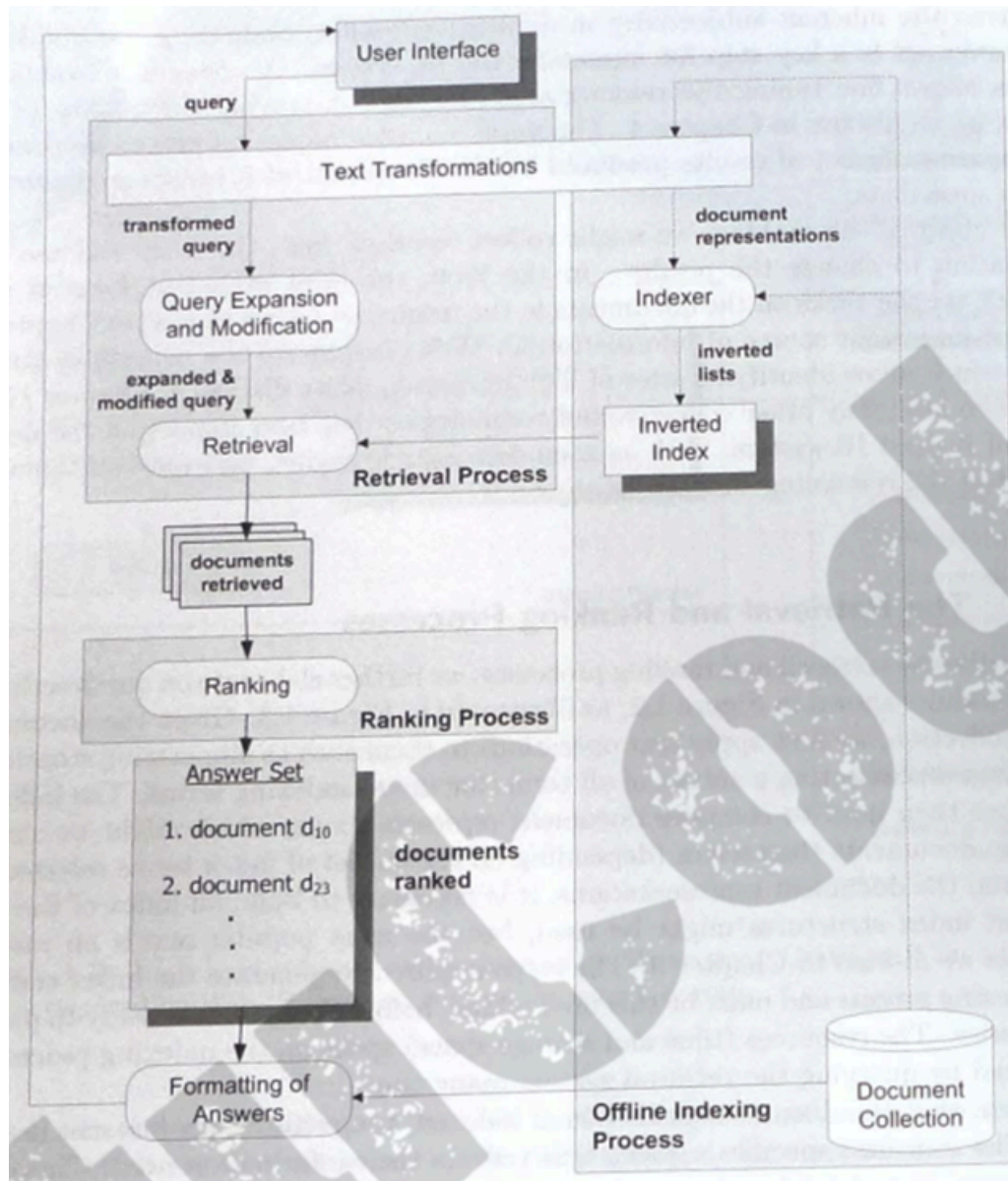
-Fast query processing is made possible by the index structure previously built.

Ranking Process

-The retrieved documents are ranked according to a likelihood of relevance to the user.

-The top ranked documents are then formatted for presentation to the user.

-The formatting consists of retrieving the title of the documents and generating snippets for them



Question 2

a. (i). Query Specification

• Methods:

- Users primarily specify queries by entering keywords into a search box.
- Selecting links from directories or other information displays.

• Characteristics:

- Web queries are typically short, often consisting of just one to three words.
- "Testing the Waters": Users often start with short queries to explore the search results and refine their search terms iteratively.
- "Orienteering Strategy": Users often use a general query to find promising websites and then navigate within those sites to find more specific information.

• Interfaces:

- The standard interface is a text box for entering search terms.

- Some search engines use forms with multiple fields for more complex query specifications.
- **Boolean Operators and Command-Line Syntax:**
 - Limited Use: While supported by some search engines, Boolean operators and command-line syntax are not widely used by most web users due to their complexity.
 - Difficulty: Many users find these advanced syntaxes confusing and difficult to use correctly.
- **Evolution of Web Search:**
 - Early Search Engines: Initially focused on Boolean operators and command-line syntax, which were often difficult for users.
 - Shift to Keyword Queries: Web search engines shifted towards keyword-based queries, with a focus on ranking results based on the relevance of keywords within the content.
 - Conjunctive Queries: Google introduced conjunctive queries, requiring all search terms to be present in the results, which improved search accuracy.
 - Sophistication: Modern search engines use more sophisticated ranking algorithms that consider various factors like term proximity, page importance, and user intent.

(ii) Query Reformulation:

-After a query is specified and results have been produced, a number of tools exist to **help the user reformulate their query**, or take their information seeking process in a new direction.

-One of the most important query reformulation techniques consist of,
 >Showing terms related to the query or to the documents retrieved in response to the query.

-A special case of this is **spelling corrections** or **suggestions**; by one estimate typographical errors occur in about 10-15% of queries.

In search interfaces, usually only one suggested alternative is shown; clicking on that alternative **re-executes the query**.

-In addition to spelling suggestions, search interfaces are increasingly employing **related term suggestions**, a technique often referred to as *term expansion*.

>Early attempts to use term expansions in search interfaces showed more than a dozen thesaurus terms, and often forced the user to select among these before showing any search results.

>More recent studies suggest that a smaller number of suggestions requiring only one click is a preferred approach.

-Many strategies are employed for Query term suggestions:-

- >One strategy is to base the suggestions on the **entire search of the particular user**.
- >Another strategy is to show **similar queries by other users**.

>**Relevance feedback** is another method that can be used.

The main idea is to have the user indicate which documents are relevant to their query

b. Visualization of search interface:

-Text as a representation is highly effective for conveying abstract information, but reading and even scanning text is a highly cognitively taxing activity, and must be done in a linear fashion.

-By contrast images can be scanned quickly and the visual system perceives information in parallel.

-People are highly attuned to images and visual information.

-A visual representation can communicate some kinds of information much more rapidly and effectively than any other method.

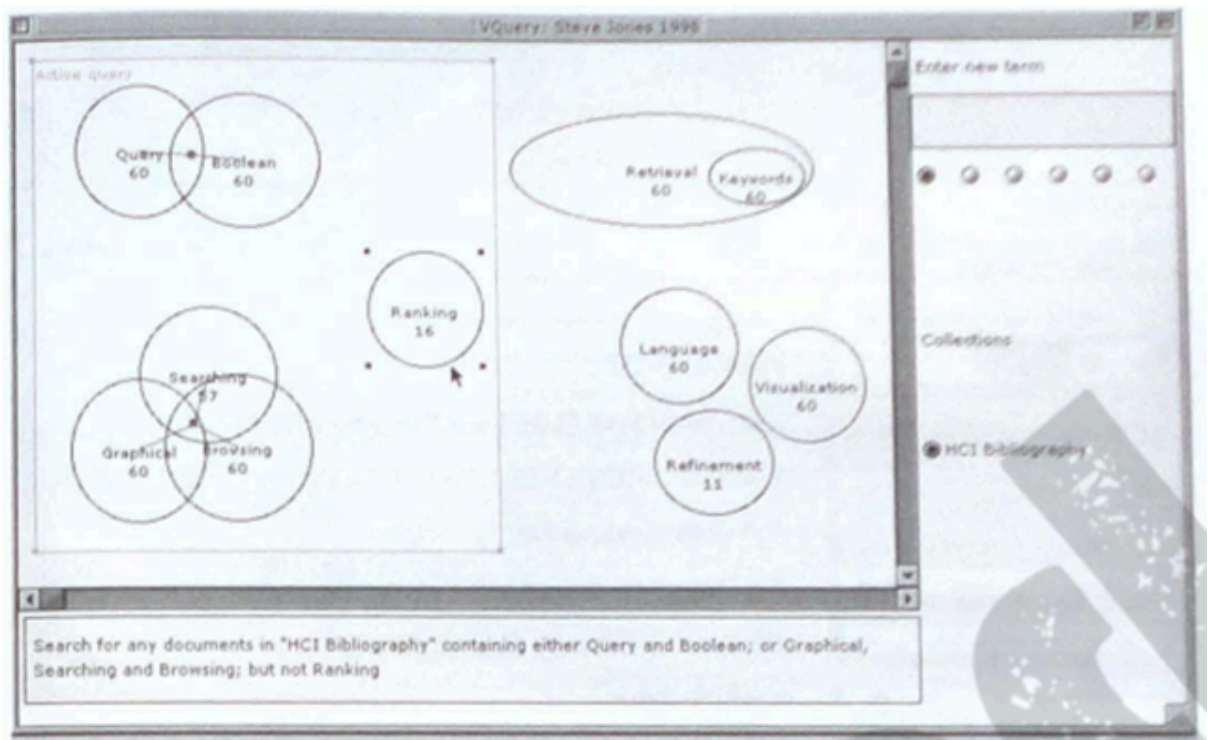
Experimentation with visualization for search has been primarily applied in the following ways:-

- Visualizing Boolean Syntax
- Visualizing Query Terms within Retrieval Results
- Visualizing Relationships among Words and Documents
- Visualization for Text Mining.

Visualizing Boolean Syntax

-A common approach to visualize Boolean Query Specification is to show Venn Diagram visually.

-A more flexible version of this is seen in the VQuery System.



-Each Query term is represented by a circle or oval,

-The Intersection among circles indicates ANDing (conjoining) of terms

- The Disjunction is represented by sets of circles within an active area of the canvas
- And Negation is done by deselecting a circle within the active area.
- One Problem with Boolean queries is that they can easily end up with empty results or too many results.

To remedy this, the filter-flow visualization allows users to lay out the different components of the query, and show via a graphical flow how many hits would result after each operator is applied.

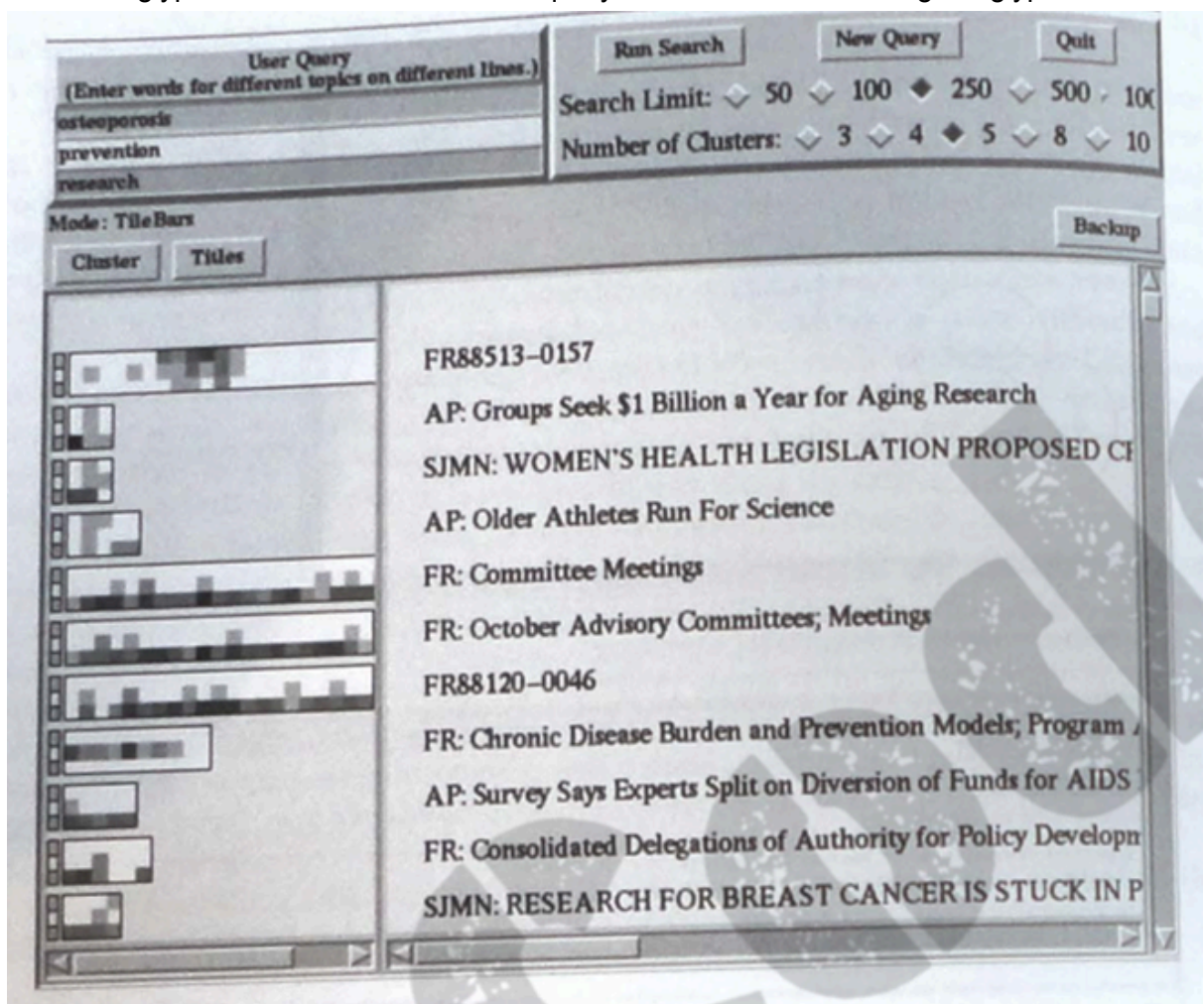
- Other types of Boolean queries include lining up blocks vertically and horizontally, and representing components of queries as overlapping "magic" lenses.

Visualizing Query Terms within Retrieval Results

- We have seen that, Query terms within the retrieved documents can help with the assessment of relevance.

- Experimental visualizations have been designed that make this relationship more explicit.

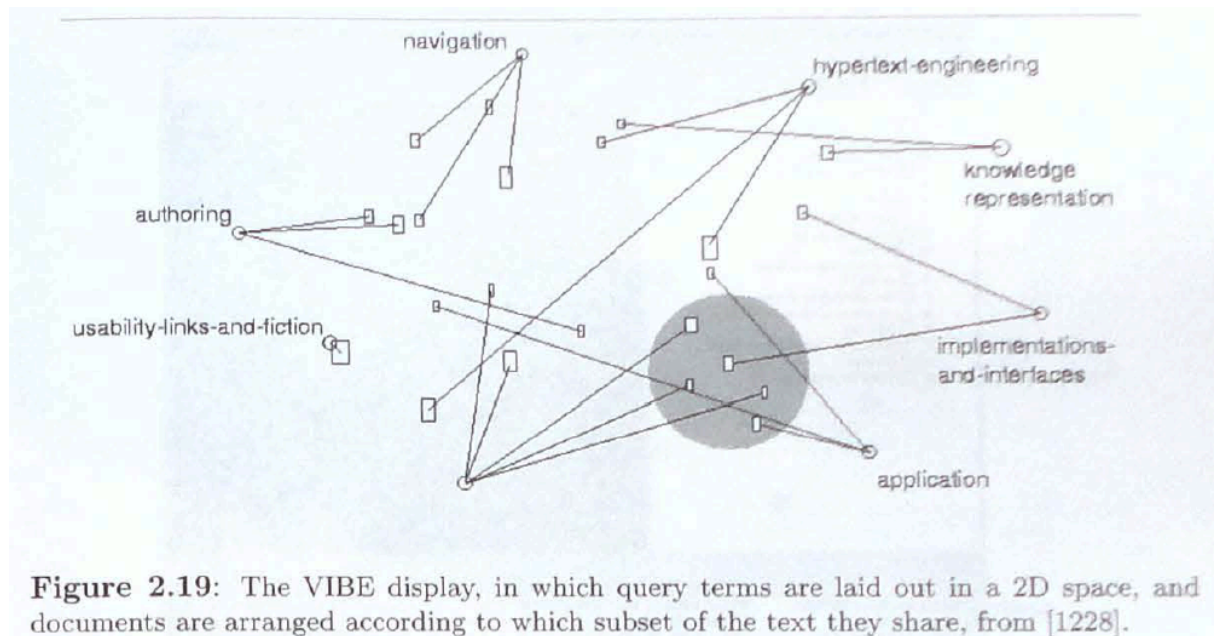
- One of the best known is the TileBars interface, in which documents are shown as horizontal glyphs with the locations of the query terms hits marked along the glyph.



- The user is encouraged to break the query into its different facets, with one concept per line, and then the horizontal rows within each document's representation show the frequency of occurrence of query terms within each topic.
 - Longer documents are divided into subtopic segments, either using paragraph or section breaks, or an automated discourse segmentation technique called TextTiling
 - Grayscale implies the frequency of query term occurrences.
- The visualization shows where the discussion of the different query topics overlaps within the document.
- Other approaches to showing query terms hits within document collections include placing the query terms in bar charts, scatter plots and tables.
 - Another variation on the idea of showing query term hits within the documents is to show thumbnails - miniaturized rendered versions of the visual appearance of the documents:-
- >Some studies show that thumbnails had no effect
 - >While a related study shows that highlighting related terms within a thumbnail did improve usability for search results

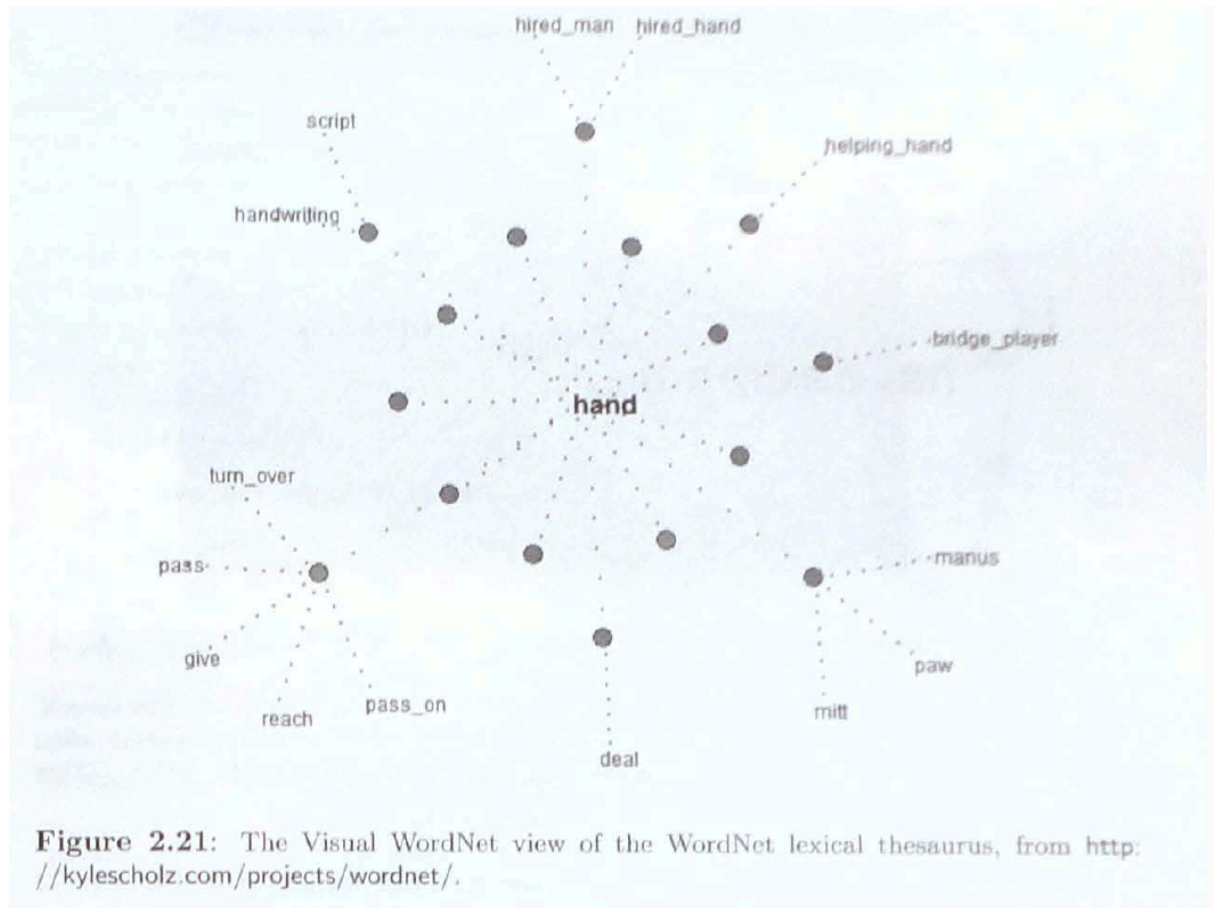
Visualizing Relationships Among Words and Documents

- Numerous visualization developers have proposed variations on the idea of placing words and documents on a two-dimensional canvas, Where Proximity of glyphs represents semantic relationships among the terms or documents.
- An Early version of this idea is seen in the VIBE interface, where Queries are laid out on a plane, and documents that contain combinations of the queries are placed midway between the icons representing those terms.



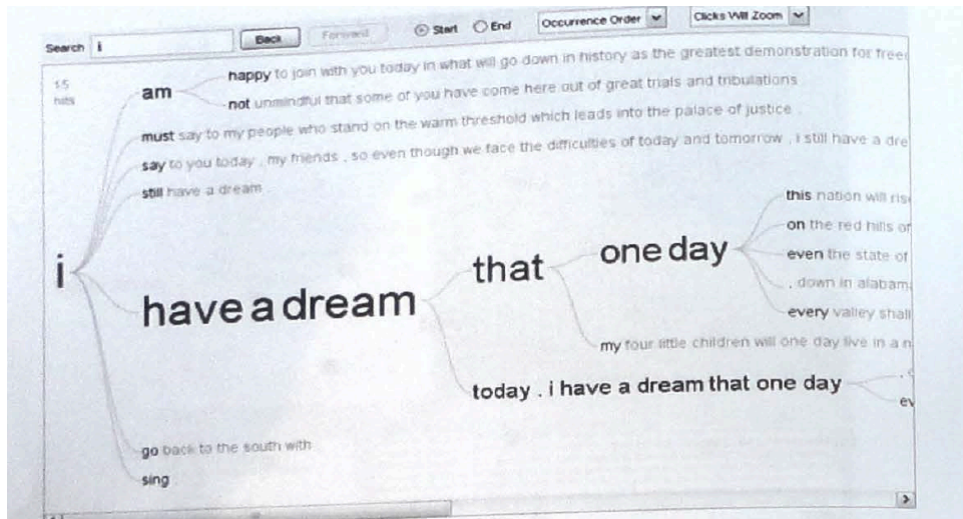
- Another variation of this idea is to map documents or words from a very high dimensional space down into a two-dimensional plane, and show where the documents or words fall within that plane.

- These views are relatively easy to compute and can be visually striking.
- However, evaluations conducted so far provide negative evidence as to their usefulness.
- A more promising application of this idea is in the layout of thesaurus terms in a small network graph, such as used in Visual Wordnet



Visualizing for Text Mining

- Visualization is not very useful for users of search systems.
- Users of search systems are not interested in seeing HOW words are distributed across documents or in viewing the most common words within a collection.
- These are interesting activities for computational linguists, analysts and curious word enthusiasts.
- Visualizations such as the Word Tree show a piece of a text concordance, allowing the user to view which words and phrases commonly precede or follow a given word.



-Visualization is also used in search interfaces intended for analysts.

Module 2

Question 3

a. A Taxonomy of IR Models

-We distinguish IR models into three major types:

>Those based on **Text**:-

Here we distinguish into models for unstructured text and models that structure of the text

i) Unstructured text: where text is modelled as simply a sequence of words. These have three classic models in IR called **Boolean**, **vector** and **probabilistic**.

+In the **Boolean Model**, documents are represented as sets of index terms. This model is called as *Set Theoretic*

+In the **Vector Model**, documents and queries are represented as vectors in a t-dimensional space. The model is *algebraic*.

+In the **Probabilistic Model**, The framework for modelling document and query representation is based on probability theory. Thus the model is *Probabilistic*.

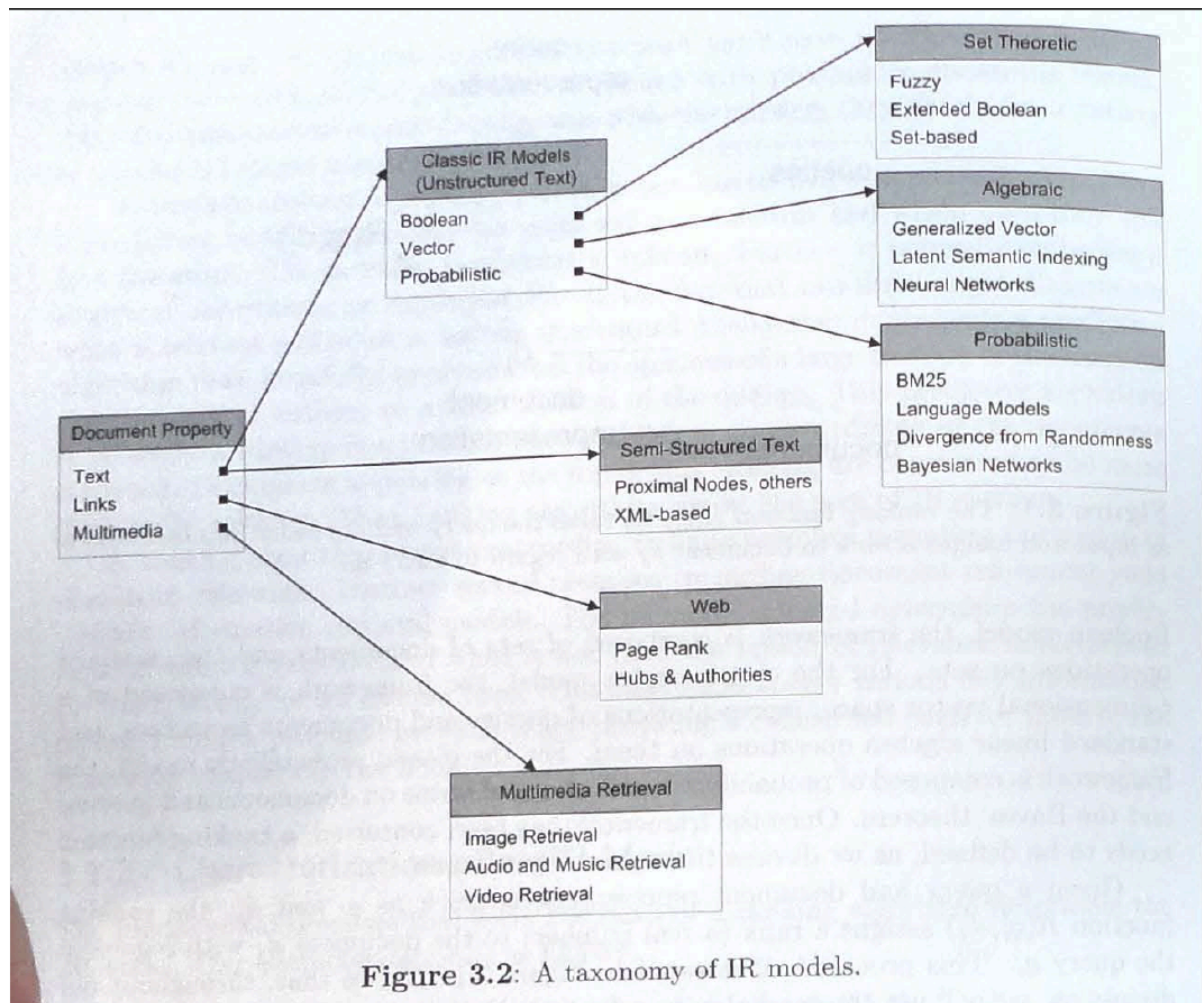


Figure 3.2: A taxonomy of IR models.

ii) Semi-structured text: Structural components of the text such as title, sections, subsections and paragraphs are an integral part of the model. We consider models that deal with structure provided within the text, we consider indexing approaches such as the **Proximal nodes** and **XML-Based Indexing** Methods.

>Those based on **Links**:-

On the web, text-based ranking is not enough, it is also necessary to consider the links among webpages as an integral part of the model

This leads to link based retrieval methods, particularly **PageRank** and **Hubs** and **Authorities**

>And those based on **Multimedia Objects**:-

`The simplest form of multimedia retrieval is **image retrieval** because an image is static

`In the case of **audio** and **video**, the representation of the multimedia object has to also include the time dimensions which makes the files larger and the problem more difficult.

b. (i) The boolean Model:

-The Boolean Model is a simple retrieval model based on **set theory** and **Boolean algebra**.

-As a result, the model is quite intuitive and has **precise semantics**.

Definition In the Boolean model, all elements of the term-document matrix are either 1, to indicate presence of the term in the document, or 0, to indicate absence of the term in the document. A query q is a Boolean expression on the index terms such as, for instance, $[q = k_a \wedge (k_b \vee \neg k_c)]$. Given the query, a term conjunctive component that satisfies its conditions is called a query conjunctive component $c(q)$. By compiling all query conjunctive components, we can rewrite the query as a disjunction of those components. This is called the query disjunctive normal form, which we refer to as q_{DNF} .

-The Boolean model considers that index terms are present or absent in a document i.e, the term-document frequencies in the term-document matrix are all binary.

-A query q is composed of index terms linked by three connectives : *not, and, or*

-If the document satisfies the conditions involving the query terms, then there is a query conjunctive component that matches the **document conjunctive component**.

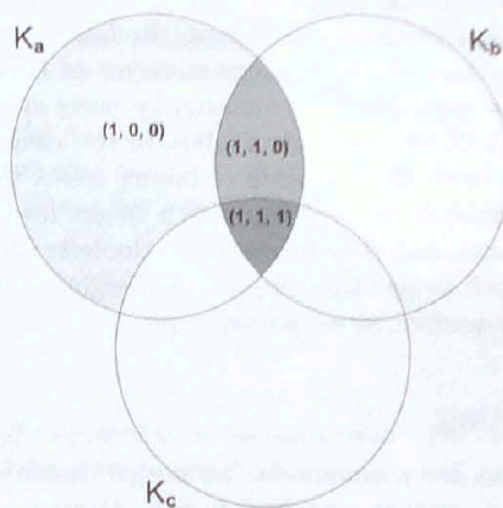


Figure 3.4: The three conjunctive components for the query $[q = k_a \wedge (k_b \vee \neg k_c)]$.

represented in disjunctive normal form as

$$q_{DNF} = (1, 1, 1, 0) \vee (1, 1, 1, 1) \vee (1, 1, 0, 0) \vee (1, 1, 0, 1) \vee (1, 0, 0, 0) \vee (1, 0, 0, 1)$$

Definition In the Boolean model, a query q is a conventional Boolean expression on index terms. Let $c(q)$ be any of the query conjunctive components. Given a document d_j , let $c(d_j)$ be the corresponding document conjunctive component. Then, the similarity of the document d_j to the query q is defined as

$$\text{sim}(d_j, q) = \begin{cases} 1 & \text{if } \exists c(q) \mid c(q) = c(d_j) \\ 0 & \text{otherwise} \end{cases} \quad (3.1)$$

If $\text{sim}(d_j, q) = 1$ then the Boolean model predicts that the document d_j is relevant to the query q (it might not be). Otherwise, the prediction is that the document is not relevant.

Advantages:-

- +The main *advantages* of the Boolean model are the **clean formalism** behind the model and
- +its **simplicity**, with the adoption of binary index term weights

Disadvantages:-

- The main *disadvantages* are that there is no **ranking**, which might lead to the retrieval of too few or too many documents
- Lack of **Index term weighting**,
- It does not take into account Term-term correlation and assumes that the terms are **mutually independent**.

Example:

Document	Content
D1	Computer Information Retrieval
D2	Computer Retrieval
D3	Information
D4	Computer Information

Step 2: Formation of Term × Document Matrix

Keywrds\Documents	D1	D2	D3	D4
Computer (K1)	1	1	0	1
Information (K2)	1	0	1	1
Retrieval (K3)	1	1	0	0

Step 3: Check user query is in DNF form. If not convert to DNF

$$q = \text{information} \wedge \text{retrieval}$$

$$\overline{q_{dnf}} = \text{information} \wedge \text{retrieval}$$

$$q_{cc} = \text{information} \wedge \text{retrieval}$$

Step 4: Find similaity between query and each document by applying similairt function

$q_{cc} = \text{information} \wedge \text{retrieval}$	K2	K3
D1	1	1
D2	0	1
D3	1	0
D4	1	0

Two keywords

First is k2 and second is K3

For query, $g_i(q_{cc}) = (1,1)$ as they are present

Only for D1, $g_i(D1) = (1,1)$ as they are present

Documents D1 gets retrieved

(ii) Term Weighing:

-We notice that not all terms are equally useful in describing a **documents contents**.

-Distinct index terms have varying importance when used to describe document contents.

This effect is captured through the assignment of **numerical weights** to each **index term** of a document

Definition To characterize term importance, a weight $w_{i,j}$, $w_{i,j} > 0$, is associated with each index term k_i of a document d_j in the collection. For an index term k_i that does not appear in the document, $w_{i,j} = 0$.

-The **weights** are influenced by the **documents in the collection**

-To account for term importance we must compute **weights** that reflect the importance of the term in **the collection** and in **each** particular **document**.

These weights depend on the frequencies of occurrence of terms within documents, which we define as follows.

Definition Let $f_{i,j}$ be the frequency of occurrence of index term k_i in the document d_j , i.e., the number of times that term k_i appears in the text of document d_j . The total frequency F_i of term k_i in the collection is the sum of the frequencies of occurrence of the term in all documents, i.e.,

$$F_i = \sum_{j=1}^N f_{i,j} \quad (3.2)$$

where N is the number of documents in the collection. The document frequency of a term k_i is the number of documents in which it occurs and is indicated simply as n_i . Notice that $n_i \leq F_i$.

Document	Content
Doc1	Health observances for March awareness
Doc2	Health oriented calendar
Doc3	Awareness news for March awareness
Query	March calendar
Stopword	for

Sr. No.	Index Term or Keywords	Doc1	Doc2	Doc3	Query: March Calendar	Count	$\log(1+3/\text{count})$
1	Health	0.40	0.40	0.00	0.20	2	0.40
2	observances	0.60	0.00	0.00	0.30	1	0.60
3	March	0.40	0.00	0.20	0.60	2	0.40
4	Awareness	0.40	0.00	0.40	0.20	2	0.40
5	Oriented	0.00	0.60	0.00	0.30	1	0.60
6	Calendar	0.00	0.60	0.00	0.90	1	0.60
7	News	0.00	0.00	0.30	0.30	1	0.60

$$w_{i,j} = f_{i,j} \times idf_i$$



$$w_{i,q} = (0.5 + f_{i,q}) \times idf_i$$

$$\text{sim}(d_j, q) = \frac{\vec{d_j} \cdot \vec{q}}{|\vec{d_j}| \times |\vec{q}|}$$

$$\text{sim}(d_j, q) = \frac{\sum_{i=1}^t w_{i,j} \times w_{i,q}}{\sqrt{\sum_{i=1}^t w_{i,j}^2} \times \sqrt{\sum_{i=1}^t w_{i,q}^2}}$$

$$\text{sim}(\text{Doc3}, q) = \frac{0.18}{0.54 \times 1.23} = 0.27$$

Sr. No.	Index Term or Keywords	Doc3	Query	
1	Health	0.00	0.20	0.00
2	observances	0.00	0.30	0.00
3	March	0.20	0.60	0.12
4	Awareness	0.40	0.15	0.06
5	Oriented	0.00	0.30	0.00
6	Calendar	0.00	0.90	0.00
7	News	0.30	0.30	0.09
		0.54	1.23	0.18

$$\sqrt{\sum_{i=1}^t w_{i,j}^2} \quad \sqrt{\sum_{i=1}^t w_{i,q}^2} \quad \sum_{i=1}^t w_{i,j} \times w_{i,q}$$

Document	Similarity Score
Doc1	0.50
Doc2	0.7
Doc3	0.27

Question-4

- Representing documents and queries through sets of keywords yields descriptions which are only **partially related** to the **real semantic contents** of the respective documents and queries.

As a result, the matching of the documents to the query terms is approximate. This can be modelled by considering each query term defines a **fuzzy set** and that each document has a **degree of membership** (usually smaller than 1) in this set.

Fuzzy Set Theory

Fuzzy set theory [1763] deals with the representation of classes whose boundaries are not well defined. The key idea is to associate a membership function with the elements of the class. This function takes values in the interval $[0,1]$ with 0 corresponding to no membership in the class and 1 corresponding to full membership. Membership values between 0 and 1 indicate *marginal* elements of the class. Thus, membership in a fuzzy set is a notion intrinsically *gradual* instead of abrupt (as in conventional Boolean logic).

Definition A fuzzy subset A of a universe of discourse U is characterized by a membership function $\mu_A : U \rightarrow [0,1]$ which associates with each element u of U a number $\mu_A(u)$ in the interval $[0,1]$.

The three most commonly used operations on fuzzy sets are:

- >The **complement** of a fuzzy set
- >The **union** of two or more fuzzy sets

>The **intersection** of two or more fuzzy sets

Definition Let U be the universe of discourse, A and B be two fuzzy subsets of U , and \bar{A} be the complement of A relative to U . Also, let u be an element of U . Then,

$$\mu_{\bar{A}}(u) = 1 - \mu_A(u) \quad (3.27)$$

$$\mu_{A \cup B}(u) = \max(\mu_A(u), \mu_B(u)) \quad (3.28)$$

$$\mu_{A \cap B}(u) = \min(\mu_A(u), \mu_B(u)) \quad (3.29)$$

Fuzzy sets are useful for representing vagueness and imprecision and have been applied to various domains. In what follows, we discuss their application to information retrieval.

information retrieval problem in terms of fuzzy sets as follows.

A thesaurus can be constructed by defining a *term-term correlation matrix* \mathbf{C} (called *keyword connection matrix* in [1224]) whose rows and columns are associated with the index terms in the document collection. This is exactly the term-term correlation matrix defined in section 3.2.3, but with different weights for the correlations, as follows. In this case, the matrix \mathbf{C} , a normalized correlation factor $c_{i,\ell}$ between two index terms k_i and k_ℓ is defined as

$$c_{i,\ell} = \frac{n_{i,\ell}}{n_i + n_\ell - n_{i,\ell}} \quad (3.30)$$

where n_i is the number of documents which contain the index term k_i , n_ℓ is the number of documents which contain the index term k_ℓ , and $n_{i,\ell}$ is the number of documents which contain both index terms. Such a correlation metric is quite common and has been used extensively with clustering algorithms as detailed in Chapter 8.

We can use the term correlation matrix \mathbf{C} to associate a fuzzy set with each index term k_i . In this fuzzy set, a document d_j has a degree of membership $\mu_{i,j}$ given by

$$\mu_{i,j} = 1 - \prod_{k_\ell \in d_j} (1 - c_{i,\ell}) \quad (3.31)$$

which computes an algebraic sum (here implemented as the complement of a negated algebraic product) over all index terms in the document d_j . A document d_j belongs to the fuzzy set associated with index term k_i if its own terms are related to k_i . Whenever

Example:

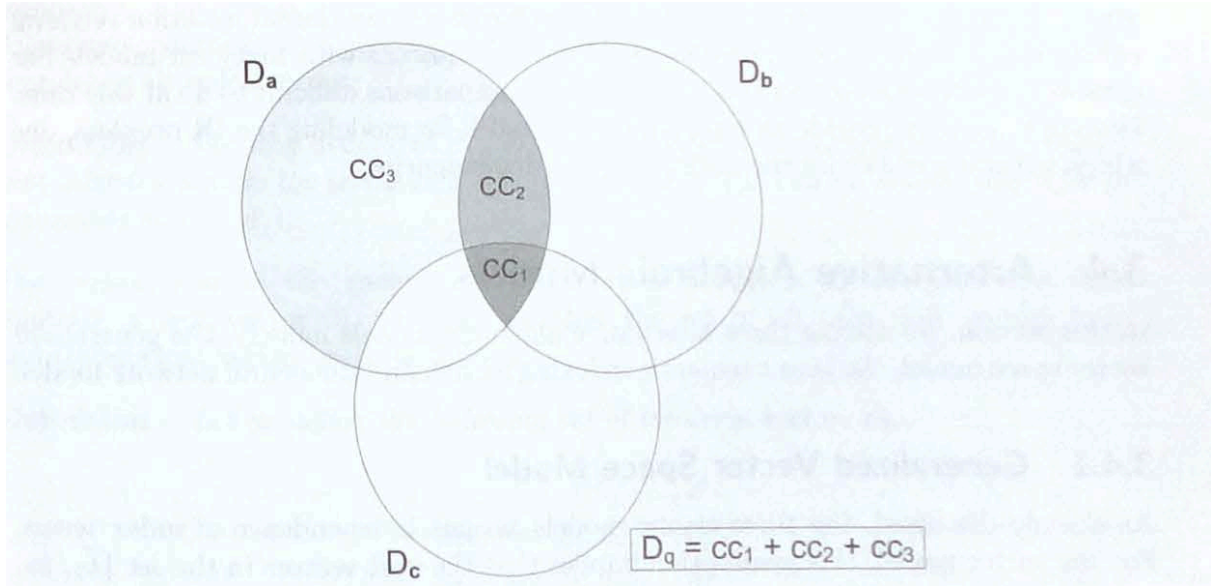


Figure 3.12: Fuzzy document sets for the query $[q = k_a \wedge (k_b \vee \neg k_c)]$. Each cc_i , $i \in \{1, 2, 3\}$, is a conjunctive component. D_q is the query fuzzy set.

the fuzzy set D_q can be computed as follows.

$$\begin{aligned}
 \mu_{q,j} &= \mu_{cc_1+cc_2+cc_3,j} \\
 &= 1 - \prod_{i=1}^3 (1 - \mu_{cc_i,j}) \\
 &= 1 - (1 - \mu_{a,j} \mu_{b,j} \mu_{c,j}) \times (1 - \mu_{a,j} \mu_{b,j} (1 - \mu_{c,j})) \times \\
 &\quad (1 - \mu_{a,j} (1 - \mu_{b,j}) (1 - \mu_{c,j}))
 \end{aligned}$$

Document	Content
D1	Shipment of gold damaged in a fire
D2	Delivery of silver arrived in a silver truck
D3	Shipment of gold arrived in a truck
Query	gold silver truck
Keywords	shipment, gold, damaged, fire, delivery, silver, arrived, truck,

	Gold	Silver	Truck
Arrived	1/3	1/2	2/2
Damaged	1/2	0	0
Delivery	0	1	1/2
Fire	1/2	0	0
Gold	1	0	1/3
Silver	0	1	1/2
Shipment	1	0	1/3
Truck	1/3	1/2	1

- The correlation factor $c_{i,l}$ can be used to define fuzzy set membership for a document d_j as follows:

$$\mu_{i,j} = 1 - \prod_{k_l \in d_j} (1 - c_{i,l})$$

The above expression computes an algebraic sum over all terms in d_j

$(1 - c_{i,l})$ is the fraction of documents containing one of term i and term l but not both

Document 1

$c_{i,l}$

Query(i)\Keywords(l)	shipment	gold	damaged	fire
gold	1	1	1/2	1/2
silver	0	0	0	0
truck	1/3	1/3	0	0

Query (i)\Keywords(l)	shipment	gold	damaged	fire	$\prod_{k_l \in d_j}$	$1 - \prod_{k_l \in d_j} (1 - c_{i,l})$
gold	(1-1)	(1-1)	(1-1/2)	(1-1/2)	0	1
silver	(1-0)	(1-0)	(1-0)	(1-0)	1	0
truck	(1-1/3)	(1-1/3)	(1-0)	(1-0)	4/9	5/9

Documents\Query	gold	silver	truck
D1	1	0	5/9
D2	5/9	1	1
D3	1	3/4	1

Document	Content	Documents\Query	gold	silver	truck
D1	Shipment of gold damaged in a fire	D1	1	0	5/9
D2	Delivery of silver arrived in a silver truck	D2	5/9	1	1
D3	Shipment of gold arrived in a truck	D3	1	3/4	1

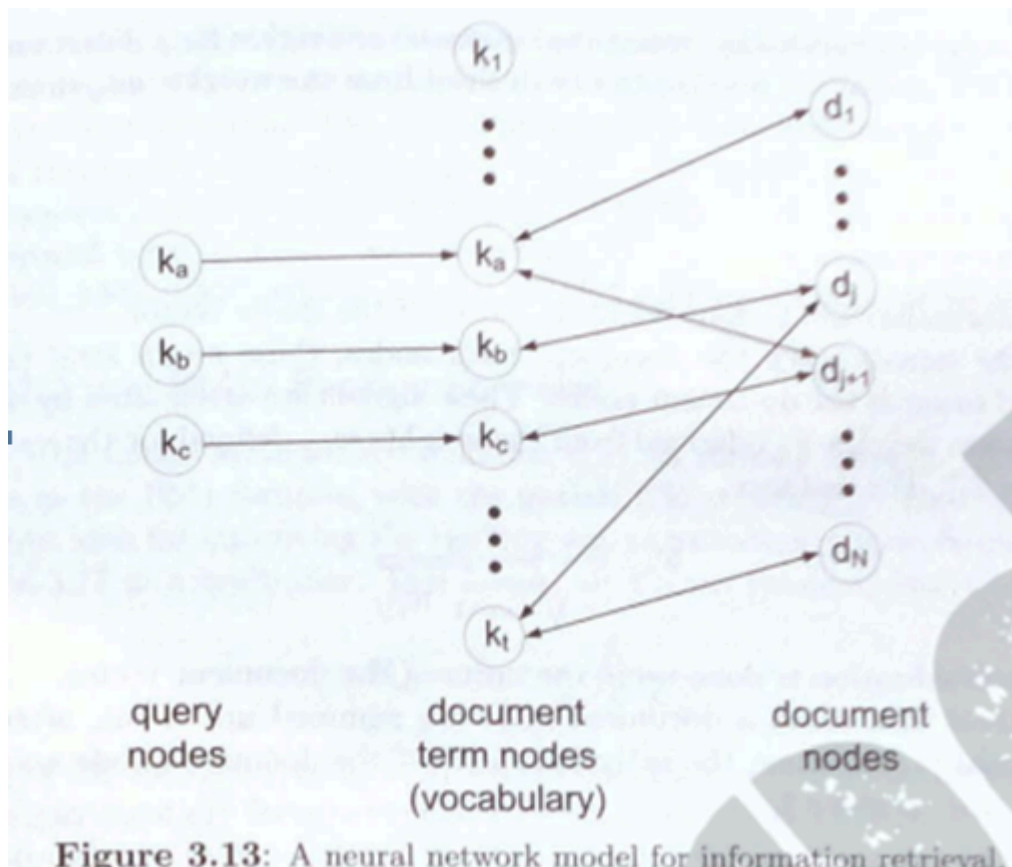
1) Query: Gold AND Silver Use Min operator (0,5/9,3/4) Ranking D3,D2,D1	2) Query: Silver AND Truck Use Min operator (0,1,3/4) Ranking D2,D3,D1
3) Query: Gold AND Truck Use Min operator (5/9, 5/9,1) Ranking D3,D1,D2	4) Query: Gold AND Silver AND Truck Use Min operator (0,,5/9,3/4) Ranking D3,D2,D1
5) Query: Gold OR Silver Use Max operator (1,1,1) Ranking D1,D2,D3	6) Query: Gold AND (Silver OR Truck) (Gold AND Silver) OR (Gold AND Truck)

b. (i) Neural Network Model:

-In an information retrieval system, **index terms** in documents and queries **have to be matched** and **weighted** for computation of a ranking.

-Since neural networks are known to be good pattern matchers, it is natural to consider their usage as an **alternative model** for **information retrieval**

-A **neural network** is an oversimplified graph representation of the mesh of interconnected neurons in a human brain.



- A **weight** is assigned to each edge of our neural network.
- At each instant the state of a node is defined by its **activation level**.

We first observe that the neural network in Figure 3.13 is composed of three layers: one for the query terms, one for the document terms, and a third one for the documents themselves. Observe the similarity between the topology of this neural network and the topology of the inference network depicted in Figure 3.15. Here, however, the query term nodes are the ones which initiate the inference process by sending signals to the document term nodes. Following that, the document term nodes might themselves generate signals to the document nodes. This completes a first phase in which a signal travels from the query term nodes to the document nodes (i.e., from the left to the right hand side in Figure 3.13).

The neural network, however, does not stop after the first phase of signal propagation. The document nodes might, in their turn, generate new signals which are directed back to the document term nodes (this is the reason for the bidirectional edges between document term nodes and document nodes). Upon receiving this stimulus, the document term nodes might again fire new signals directed to the document nodes, repeating the process. The signals become weaker at each iteration and the spread activation process eventually halts. This process might activate a document d_i even when such a document does not contain any query terms. Thus, the whole process can be interpreted as the activation of a built-in thesaurus.

D1. Computer Information Retrieval

D2. Computer Retrieval

D3. Information

D4. Computer Information

Query: Information Retrieval

Sr. No.	Index Term or Keywords	Doc1	Doc2	Doc3	Doc4	Count(ni)	Query: Information, Retrieval
1	Computer	1	1	0	1	3	0
2	Information	1	0	1	1	3	1
3	Retrieval	1	1	0	0	2	1
Max Frequency		1	1	1	1		1

N=4

Sr. No.	Index Term or Keywords	Doc1	Doc2	Doc3	Doc4	Count (ni)	Query: Information, Retrieval	idf=log(1+N/ni)	$w_{i,q}$	Normalized $w_{i,q}$
1	Computer	1	1	0	1	3	0	0.37	0.18	0.20
2	Information	1	0	1	1	3	1	0.37	0.55	0.60
3	Retrieval	1	1	0	0	2	1	0.48	0.72	0.78
Max Frequency		1	1	1	1		1			

$$w_{i,q} = \left(0.5 + \frac{freq_{i,q}}{max freq_{i,q}}\right) \times \log \frac{N}{n_i}$$

$$\bar{w}_{i,q} = \frac{w_{i,q}}{\sqrt{\sum_{i=1}^t w_{i,q}^2}}$$

Sr. No.	Index Term or Keywords	Doc1	Doc2	Doc3	Doc4	idf=log(1+N/ni)
1	Computer	1	1	0	1	0.37
2	Information	1	0	1	1	0.37
3	Retrieval	1	1	0	0	0.48
Max Frequency		1	1	1	1	

$$w_{i,j} = f_{i,j} \times \log \frac{N}{n_i}$$

$$\bar{w}_{i,j} = \frac{w_{i,j}}{\sqrt{\sum_{i=1}^t w_{i,j}^2}}$$

Sr. No.	Index Term or Keywords	Doc1	Doc2	Doc3	Doc4	$w_{i,(j=1)}$	$w_{i,(j=2)}$	$w_{i,(j=3)}$	$w_{i,(j=4)}$
1	Computer	0.37	0.37	0.00	0.37	0.52	0.61	0.00	0.71
2	Information	0.37	0.00	0.37	0.37	0.52	0.00	1.00	0.71
3	Retrieval	0.48	0.48	0.00	0.00	0.68	0.79	0.00	0.00

$$\sum_{i=1}^t w_{i,q} w_{i,j} = \frac{\sum_{i=1}^t w_{i,q} \times w_{i,j}}{\sqrt{\sum_{i=1}^t w_{i,q}^2} \times \sqrt{\sum_{i=1}^t w_{i,j}^2}}$$

Sr. No.	Index Term or Keywords	Doc1	Query	Weight
1	Computer	0.52	0.20	0.10
2	Information	0.52	0.60	0.31
3	Retrieval	0.68	0.78	0.52
			sum=	0.94
	Norm=	1	1	
			Weight	0.94

Sr. No.	Index Term or Keywords	Doc2	Query	Weight
1	Computer	0.61	0.20	0.12
2	Information	0.00	0.60	0.00
3	Retrieval	0.79	0.78	0.61
			sum=	0.74
	Norm=	1	1	0.74
			Weight	0.74

Query: Information Retrieval

Document No.	Weight
Doc1	0.94
Doc2	0.74
Doc3	0.74
Doc4	0.56

(ii) Latent Semantic Indexing Model:

-The main idea in the *latent semantic indexing model* is to map each document and query vector into a **dimensional space** composed of **concepts**.

-This is accomplished by first mapping the **index term** into this **dimensional space of concepts** and then **modelling documents** and queries in terms of these mappings.

-The claim is that retrieval in the reduced **space of concepts** may **be superior** to retrieval in the space of **index terms**.

-The latent semantic indexing model introduces an interesting conceptualization of the information retrieval problem based on SINGULAR VALUE DECOMPOSITION.

-From a practical point of view the **latent semantic indexing model** has not yielded encouraging results.

Definition As before, let t be the number of index terms in the collection, N be the total number of documents, and $M = [m_{ij}]$ be a term-document matrix with t rows and N columns. To each element m_{ij} of this matrix assign a weight w_{ij} associated with the term-document pair (k_i, d_j) . This w_{ij} weight can be generated using the TF-IDF weights of the classic vector model.

Latent semantic indexing proposes to decompose the M matrix in three components using singular value decomposition, as follows.

$$M = K \cdot S \cdot D^T$$

The matrix K is the matrix of eigenvectors derived from the term-term correlation matrix given by $C = M \cdot M^T$ (see section 3.2.3 for a description of the term-term correlation matrix). The matrix D^T is the matrix of eigenvectors derived from the transpose of the document-document matrix given by $M^T \cdot M$. The matrix S is an $r \times r$ diagonal matrix of singular values where $r = \min(t, N)$ is the rank of M .

Example:

```
M=[2 3 4 5 0 0 0 0;
    6 1 2 3 0 0 0 0;
    5 3 3 3 0 0 0 0;
    0 0 0 0 5 2 5 8;
    0 0 0 0 6 8 3 3;
    0 0 0 0 4 3 2 7];
```

size of M is 6 × 8

r (rank)=min(8,6)=6

size of S is 6 × 6

```
S =
    16.5610    0    0    0    0    0
         0   11.8379    0    0    0    0
         0    0    6.0356    0    0    0
         0    0    0    3.7986    0    0
         0    0    0    0    1.8179    0
         0    0    0    0    0    1.1984
```

D =

size of doc × concepts					
0	-0.5688	0	-0.7697	0	-0.2900
0	-0.5604	0	0.6207	0	-0.5483
0	-0.6021	0	0.1493	0	0.7844
-0.6234	0	-0.5325	0	-0.5725	0
-0.5870	0	0.8025	0	-0.1072	0
-0.5165	0	-0.2693	0	0.8128	0

K =

size of terms × concepts					
0.0000	-0.6344	-0.0000	0.7718	0.0000	0.0432
-0.0000	-0.3441	0.0000	-0.3265	0.0000	0.7800
0.0000	-0.4394	-0.0000	-0.3657	-0.0000	0.0804
-0.0000	-0.5348	0.0000	-0.4049	0.0000	-0.6191
-0.5256	0	0.1781	0	-0.1401	0
-0.4524	0	0.7533	0	0.2397	0
-0.3569	0	-0.1315	0	-0.8574	0
	0	-0.6192	0	0.4334	0

size of Concept × concepts

14

r (rank)=min(8,6)=6

size of S is 6 × 6

S =

16.5610	0	0	0	0	0
0	11.8379	0	0	0	0
0	0	6.0356	0	0	0
0	0	0	3.7986	0	0
0	0	0	0	1.8179	0
0	0	0	0	0	1.1984

In the Matrix S, select only the s largest singular values.

Keep the corresponding column in K and D^t.

The resultant matrix is called M_s and is given by

$$M_s = K_s S_s D_s^t$$

where s, s < r is the dimensionality of the concept space.

The parameter s should be

- large enough to allow fitting the characteristics of the data
- small enough to filter out the non-relevant representational details

$$\begin{aligned} \vec{M}_s^t \vec{M}_s &= (\vec{K}_s \vec{S}_s \vec{D}_s^t)^t \vec{K}_s \vec{S}_s \vec{D}_s^t \\ &= \vec{D}_s \vec{S}_s \vec{K}_s^t \vec{K}_s \vec{S}_s \vec{D}_s^t \\ &= \vec{D}_s \vec{S}_s \vec{S}_s^t \vec{D}_s^t \\ &= (\vec{D}_s \vec{S}_s)(\vec{D}_s \vec{S}_s)^t \end{aligned}$$

45.3313	44.6652	47.9853	0	0	0
44.6652	44.0088	47.2801	0	0	0
47.9853	47.2801	50.7946	0	0	0
0	0	0	106.5866	100.3638	88.3149
0	0	0	100.3638	94.5043	83.1589
0	0	0	88.3149	83.1589	73.1755

With Query

M =

1	0	0	1	0	0	0	0
2	3	4	5	0	0	0	0
6	1	2	3	0	0	0	0
5	3	3	3	0	0	0	0
0	0	0	0	5	2	5	8
0	0	0	0	6	8	3	3
0	0	0	0	4	3	2	7

S =

16.5610	0	0	0	0	0	0
0	11.8957	0	0	0	0	0
0	0	6.0356	0	0	0	0
0	0	0	3.8167	0	0	0
0	0	0	0	1.8179	0	0
0	0	0	0	0	1.3393	0
0	0	0	0	0	0	0.3618

$S_s =$

16.5610 0
0 11.8957

1.3797	7.8987	7.8053	8.3683	0	0	0
7.8987	45.2184	44.6836	47.9068	0	0	0
7.8053	44.6836	44.1551	47.3401	0	0	0
8.3683	47.9068	47.3401	50.7550	0	0	0
0	0	0	0	106.5866	100.3638	88.3149
0	0	0	0	100.3638	94.5043	83.1589
0	0	0	0	88.3149	83.1589	73.1755

The first row in the matrix $\mathbf{M}_s^t \mathbf{M}_s$ provides the ranks of all documents with respect to the query

Module 3

Question 5

a. Precision and Recall

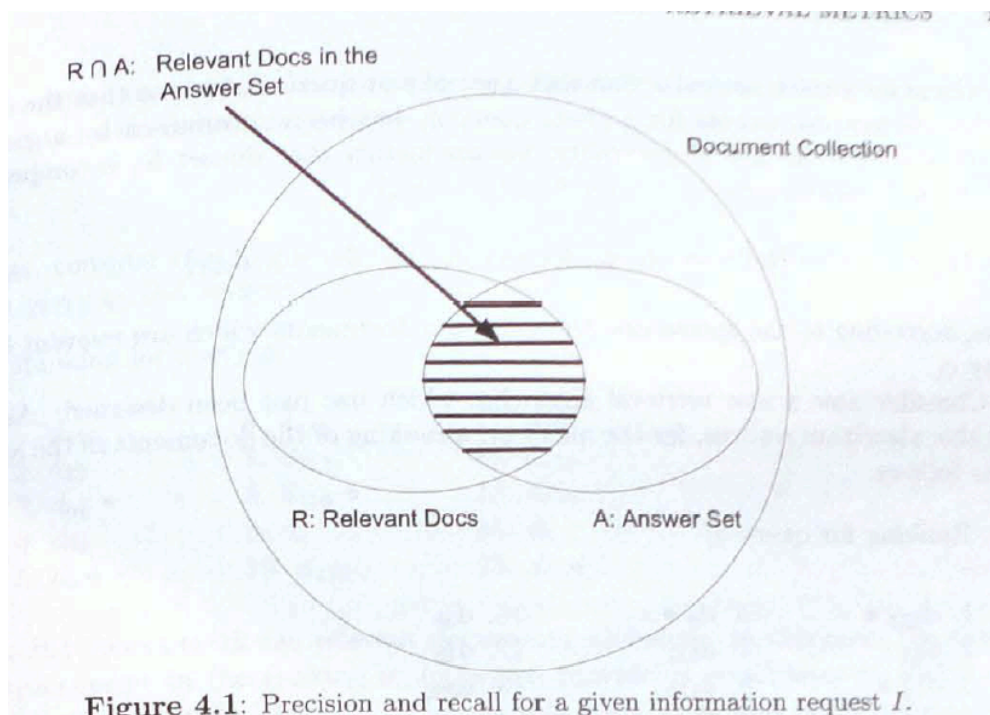


Figure 4.1: Precision and recall for a given information request I .

- Precision is the fraction of the retrieved documents (the set A) which is relevant i.e.,

$$Precision = p = \frac{|R \cap A|}{|A|} \quad (4.1)$$

- Recall is the fraction of the relevant documents (the set R) which has been retrieved i.e.,

$$Recall = r = \frac{|R \cap A|}{|R|} \quad (4.2)$$

Scenario:

- We have a collection of documents and a specific query (q1).
- A group of experts has identified 10 documents (R1) as relevant to query q1.
- $R1 = \{d3, d5, d9, d25, d39, d44, d56, d71, d89, d123\}$

Algorithm and Ranking:

- A new retrieval algorithm is introduced.
- This algorithm ranks documents in response to query q1 as follows:
 1. d123
 2. d84
 3. d56
 4. d6
 5. d8
 6. d9
 7. d511
 8. d129
 9. d187
 10. d25
 11. d38
 12. d48
 13. d250
 14. d113
 15. d3

Relevant Documents in the Ranking:

- The relevant documents (from R1) are marked with a bullet in the ranking.
- Relevant documents in the ranking: d123, d56, d9, d25, d3

Precision and Recall Calculation:

- **Precision:** The proportion of retrieved documents that are actually relevant.
- **Recall:** The proportion of relevant documents that are retrieved by the algorithm.
- **At 10% Recall (1 out of 10 relevant documents):**
 - The first retrieved document is d123, which is relevant.
 - Precision = $1/1 = 100\%$
- **At 20% Recall (2 out of 10 relevant documents):**
 - The next relevant document is d56 (ranked 3rd).
 - Precision = $2/3 = 66.6\%$
- **Precision at Higher Recall Levels:**
 - As more irrelevant documents are retrieved, precision drops.
 - If all relevant documents are not retrieved, precision eventually drops to 0%.

Example 3: Find total retrieved, total not retrieved, Precision & Recall

		Actual	
Predicted		Relevant (5)	Non Relevant (5)
	Retrieved (6)	TP(3)	FP(3)
	Not Retrieved (4)	FN(2)	TN(2)

Total Retrieved=6, , Total Not Retrieved=4

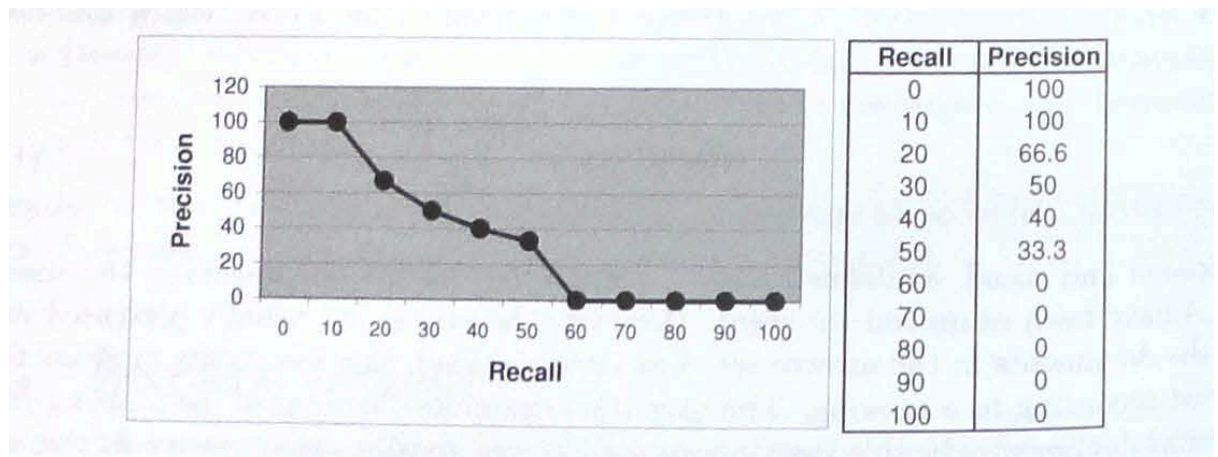
$$\text{Precision}(P) = \frac{3}{3+3} = \frac{3}{6} = 0.5$$

$$\text{Recall}(R) = \frac{3}{3+2} = \frac{3}{5} = 0.6$$

Precision-Recall Curve:

- The text mentions a precision-recall curve (Figure 4.2) that visually represents the relationship between precision and recall at different levels.
- This curve is typically used to evaluate and compare the performance of different retrieval algorithms.

In essence, this example demonstrates how to calculate precision and recall based on the ranked list of documents retrieved by an algorithm and the set of known relevant documents.



b. Explicit feedback Evaluation:

Human Experimentation in the Lab

-To Evaluate the impact of the **user interface(UI)** on **user preferences** it is necessary to run several **evaluation sessions** which look identical except for a few specific characteristics of the UI that are varied with the purpose of measuring their impact on user preferences

-Human Experimentation in the lab allows **understanding dynamic characteristics** of the user-system interaction that **cannot be evaluated** using static reference collections.

-The downside is that they are **costly to setup, costly to be repeated** and are limited to a **small set of information needs**. That are executed by a relatively small set of human subjects.

Side by Side Panels:

Side-by-Side Experiment:

- In a side-by-side experiment, users are presented with two sets of search results, typically from different search engines, side-by-side.
- The users are then asked to judge which set of results is better for a given query.
- This judgment is based on the user's assessment of the relevance and quality of the results.

Evaluation Process:

- The evaluation is based on the judgments of multiple human assessors.
- If a majority of assessors prefer one set of results over the other, it is considered to be better for that query.

A/B Testing

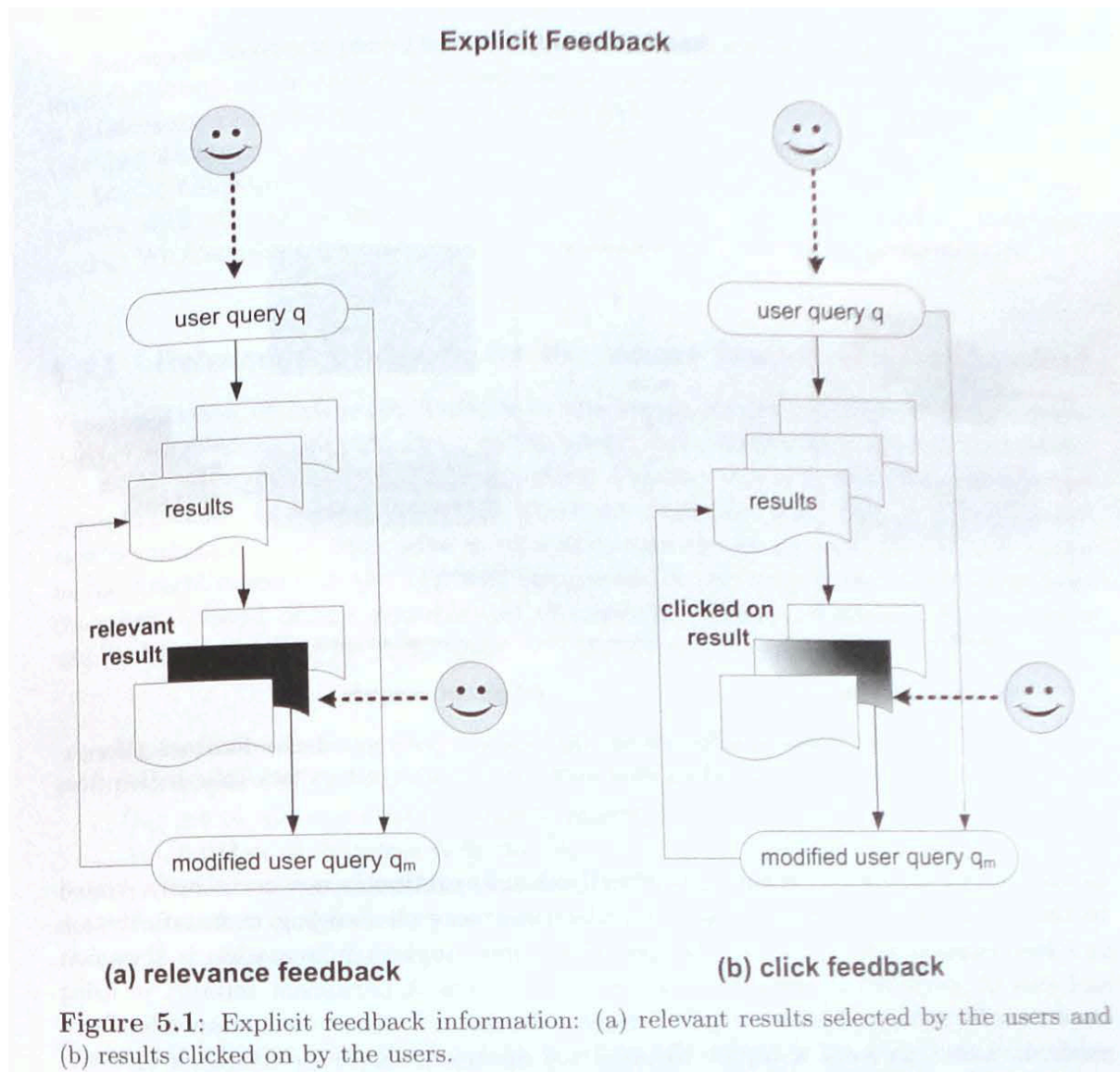
- It is also called *bucket testing*, it is a popular method for performing evaluation experiments.
- It consists of displaying to **selected users a modification** of the layout of a page
- By **analysing how the users react to the change**, it is possible to gain understanding on whether the modification is positive or not.
- Usually the set of users is a **small fraction** of all users in the site.
- The technique is **particularly important** with **heavily used sites** because a poor modification that is launched might cause an annoyance to millions

Crowdsourcing

- Recently, **crowdsourcing** has emerged as a **feasible alternative** for relevance evaluation because it combines the flexibility of the editorial approach at a larger scale.
- Crowdsourcing is a term used to describe tasks that are outsourced to a large group of people called "workers", instead of performed by an employee or contractor.
- The lower cost of running experiments makes this approach very attractive for testing new ideas with a fast turnaround

Evaluation using Clickthrough Data

- Another alternative is evaluation based on the analysis of clickthrough data.
- This is particularly attractive because clickthrough data can be collected at a very **low cost** without overhead for the user



Implicit feedback Evaluation:

Implicit Feedback Information

-In an implicit relevance feedback cycle, there is no participation of user in the feedback process. Instead the **feedback information** is **derived implicitly by the system**.

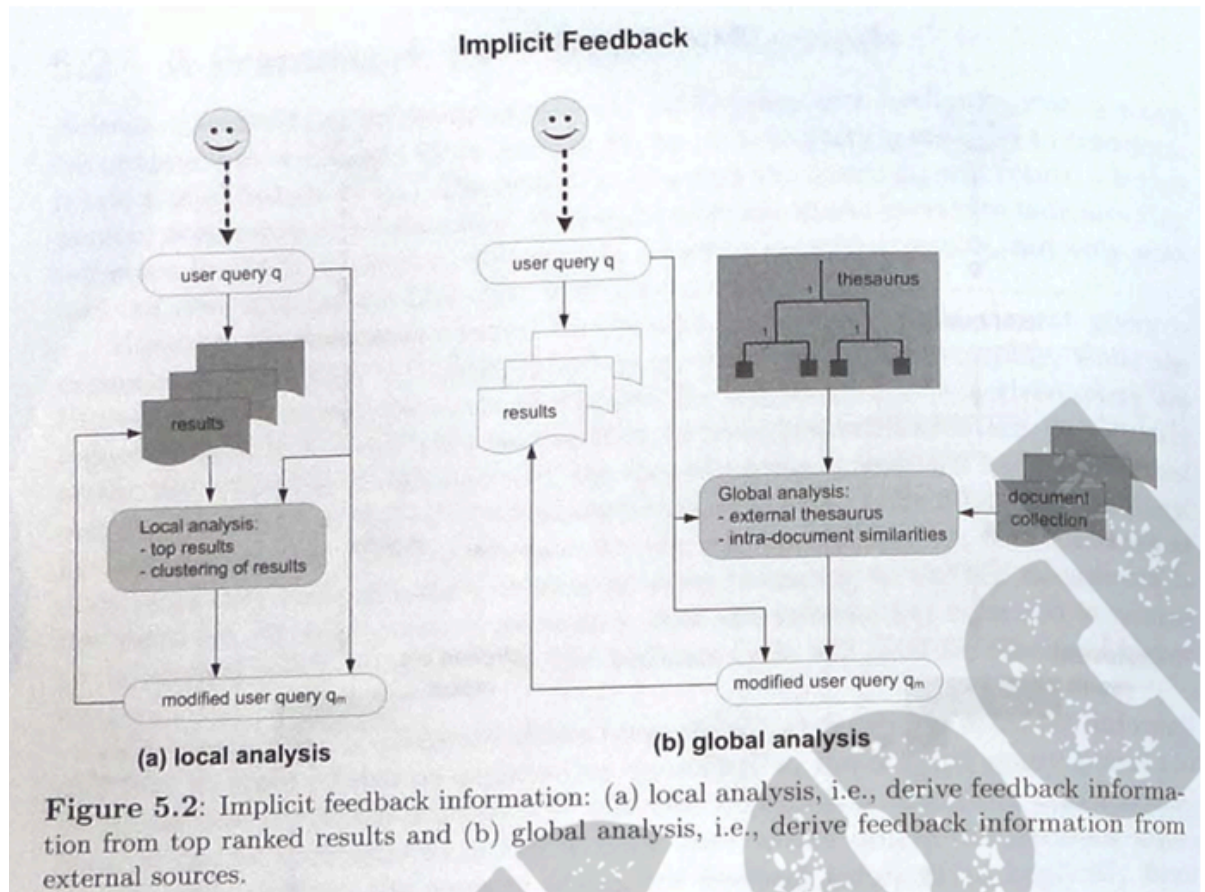
-There are two basic approaches for compiling implicit feedback information:-

a) derive the feedback information from the top ranked documents in the result set.

This is usually referred to as **local analysis**.

b) Derive the Feedback information from external sources such as a thesaurus or from term relations extracted from the document collection

This is usually referred to as **global analysis**



Question 6

a. Implicit Feedback Through Global Analysis

-The methods of local analysis extract information from the documents retrieved to expand the query.

-An **alternative approach** is to expand the query using information from the **whole set of documents in the collection**. Strategies based on this idea are called "global analysis procedures".

-There are **two modern variants** of global analysis. Both of them are based on a thesaurus-like structure built using all the documents in the collection.

Query Expansion based on a Similarity Thesaurus

-A *similarity thesaurus* is built using term-term relationships, which are derived by considering that the terms are concepts in a **concept space**.

In this concept space, **each term** is **indexed** by the documents in which it appears.

Thus the terms assume the original role of documents while documents are interpreted as indexing elements.

Given the global similarity thesaurus, query expansion is done in three steps as follows.

- First, represent the query in the same vector space used for representing the index terms.
- Second, based on the global similarity thesaurus, compute a similarity $sim(q, k_v)$ between each term k_v correlated to the query terms and the whole query q .
- Third, expand the query with the top r ranked terms according to $sim(q, k_v)$.

Step 1:

Definition With the query q is associated a vector \vec{q} given by

$$\vec{q} = \sum_{k_i \in q} w_{i,q} \vec{k}_i$$

where $w_{i,q}$ is a weight associated with the index-query pair $[k_i, q]$. This weight is given by equation 5.14, with the document d_j replaced by the query q .

Step 2:

For the second step, a similarity $sim(q, k_v)$ between each term k_v correlated to the query terms and the user query q is computed as

$$sim(q, k_v) = \vec{q} \bullet \vec{k}_v = \sum_{k_i \in q} w_{i,q} \times c_{i,v} \quad (5.16)$$

Step 3:

For the third step, the top r ranked terms according to $sim(q, k_v)$ are added to the original query q to form the expanded query q_m . To each expansion term k_v in query q_m is assigned a weight w_{v,q_m} given by

$$w_{v,q_m} = \frac{sim(q, k_v)}{\sum_{k_i \in q} w_{i,q}}$$

Query Expansion based on a Statistical Thesaurus

-The global statistical thesaurus is composed of classes that **group correlated terms** in the context of the whole collection.

-Such correlated terms can then be used to **expand the original user query**.

-To be effective, the terms selected for expansion must have **high term discrimination values**, which imply that they must be low frequency terms.

However, it is **difficult to cluster low frequency** terms due to the small amount of information about them.

-To circumvent this problem, **documents are clustered into classes** instead and the **low frequency terms** in these documents are used to define thesaurus classes using a clustering algorithm

-A document clustering algorithm that produces small and tight clusters is the *complete link algorithm*

A document clustering algorithm that produces small and tight clusters is the *complete link algorithm*, which works as follows (naive formulation).

1. Initially, place each document in a distinct cluster.
2. Compute the similarity between all pairs of clusters.
3. Determine the pair of clusters $[C_u, C_v]$ with the highest inter-cluster similarity.
4. Merge the clusters C_u and C_v .
5. Verify a stop criterion. If this criterion is not met then go back to step 2.
6. Return a hierarchy of clusters.

Example:-

Consider that the whole document collection has been clustered using the complete link algorithm. Figure 5.6 illustrates a small portion of the whole cluster hierarchy composed of clusters C_u , C_v , and C_z , for which $\text{sim}(C_u, C_v) = 0.15$ and $\text{sim}(C_{u+v}, C_z) = 0.11$, where C_{u+v} is a reference to the cluster which results from merging C_u and C_v . Notice that the similarities decrease as we move up in the hierarchy because high level clusters include more documents and thus represent a looser grouping. Thus, the tightest clusters lie at the bottom of the clustering hierarchy.

Given the document cluster hierarchy for the whole collection, the terms that compose each class of the global thesaurus are selected as follows.

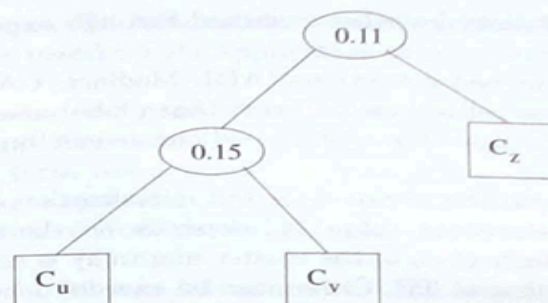


Figure 5.6: Hierarchy of three clusters generated by the complete link algorithm. The inter-cluster similarities are shown in the ovals.

b. Text Properties

(i) Information Theory

-It is difficult to formally capture how much information is encoded in a given text.

For example, a text in which one symbol **appears almost all the time** does NOT convey much information

-We can use this idea to assign different **bit sequences** or **CODES** to different symbols.

-Source Code Theorem: In an optimal encoding scheme, a symbol that is expected to occur with probability p should be assigned a codeword of length $\log_2(1/p)$ bits.

The number of bits in which a symbol is best encoded represents the **information content** of the symbol.

The average amount of information per symbol over a whole text $T = t_1 t_2 \dots t_n$ is called the *entropy* of the text under the text model used, and is given by

$$E = \frac{1}{n} \sum_{i=1}^n \log_2 \frac{1}{p_i}, \quad (6.1)$$

where p_i is the probability assigned by the model to symbol t_i . It is important to note that E is calculated from the probabilities and so it is a property of the model and not only of the text.

In the simple case where the model always assign probability p_i to the alphabet symbol s_i , the entropy can be rewritten as

$$E = \sum_{s_i \in \Sigma} p_i \log_2 \frac{1}{p_i} = - \sum_{i=1}^{\sigma} p_i \log_2 p_i \quad (6.2)$$

(ii) Modelling Natural Language

-Text is composed of symbols from a finite alphabet, which can be divided in two disjoint subsets:

- Symbols that separate words (called *separators*),
- Symbols that belong to words.

-More complex models include **finite-state models**, which define regular languages, and **grammar models** which define context-free and other languages.

However, finding the right grammar for natural language is still a difficult and ongoing problem.

-Another issue of importance is how the **different words are distributed** inside each document.

Another issue of importance is the *Zipf's Law* [1794, 649], which attempts to document. An approximate model is the *Zipf's Law* [1794, 649], which attempts to capture the distribution of the frequencies (that is, number of occurrences) of the words in the text. The rule states that the frequency f_i of the i -th most frequent word is $1/i^\alpha$ times the frequency f_1 of the most frequent word, where α is a text dependent parameter. That is,

$$f_i = \frac{f_1}{i^\alpha} \quad (6.3)$$

The original Zipf's law used $\alpha = 1$, and then when $\alpha > 1$ we name it generalized Zipf's law. For a text of n words with a vocabulary of V words, we have

$$n = \sum_{i=1}^V \frac{1}{i^\alpha} f_1 = f_1 \times \sum_{i=1}^V \frac{1}{i^\alpha}$$

The factor enclosed in brackets depends only on the text parameters α and V and is called the harmonic number $H_V(\alpha)$ of order α of V , that is

$$H_V(\alpha) = \sum_{i=1}^V \frac{1}{i^\alpha} \quad \text{and then} \quad f_1 = \frac{n}{H_V(\alpha)}. \quad (6.4)$$

An immediate consequence is that, according to this Zipfian model, the i -th most frequent word appears $n/(i^\alpha H_V(\alpha))$ times.

-A third issue is the **distribution of words** in the documents of a collection.

-The fourth issue is the **number of distinct words** in a document, this set of words is referred to as the document vocabulary V

-A last issue is the **average length** of words, this relates the text size in words with the text size in bytes.

These issues are some of the few that make modelling natural language such a difficult problem to overcome

(iii) Text Similarity

-We define notions of syntactic similar between strings or documents in the form of a **distance function**.

- **Similarity Measurement:**

- Similarity between objects is often measured using a "distance function."
- A lower distance value generally indicates higher similarity.
- Example: Hamming Distance - Measures the number of positions where two strings of the same length differ.

- **Edit Distance (Levenshtein Distance):**

- Measures the minimum number of single-character edits (insertions, deletions, substitutions) required to transform one string into another.
- Considered superior to methods like Soundex for modeling syntactic errors.
- Can be extended with weighted operations or by adding transposition as an edit operation.
- **Similarity in Documents:**
 - The concept of similarity extends to documents.
 - One approach is to compare lines in two files and compute the longest common sequence. This is used by the Unix "diff" command.
 - Limitations: Time-consuming and doesn't consider similarity within lines.
- **Other Approaches to Document Similarity:**
 - **Fingerprinting:** Extracting characteristic pieces of text to represent the document.
 - **Finding Large Repeated Pieces:** Identifying common segments within documents.
 - **Visual Display:** Tools like Dotplot create visual representations of document similarity by comparing lines and displaying the results as a heatmap.

-Examples include **Hamming Distance** and **Levenshtein Distance**(Edit distance)

-Similarity can be extended to Documents

-Most efficient document similarity measures include the use of cosine distance and the resemblance measure

The first uses the **vector model representation** of each document and apply the **cosine distance** with one of the multiple weighting functions available

The second is called **resemblance**.

-We can also transform any **similarity measure**, so long as it's in the range [0,1] into a distance function through

Another similarity measure proposed is *resemblance* [267]. If $W(d_j)$ is the set of all distinct words in document d_j , then the resemblance function $R(d_i, d_j)$ between documents d_i and d_j is defined as:

$$R(d_i, d_j) = \frac{|W(d_i) \cap W(d_j)|}{|W(d_i) \cup W(d_j)|} \quad (6.7)$$

Notice that $0 \leq R(d_i, d_j) \leq 1$ and that this measure is a particular case of the Jaccard similarity [816]. Further, we can compute resemblance using any reasonable definition of the function W . One of the most efficient techniques is shingling. Shingles are a set of contiguous terms in a document, which represent a subset of the content of the document. Using shingles to compute W , we can compute resemblance much faster as we use groups of terms instead of single words.

Notice that any similarity measure in the range $[0, 1]$ can be easily transformed in a distance function $D(d_i, d_j)$ through

$$D(d_i, d_j) = 1 - Sim(d_i, d_j) \quad (6.8)$$

Module 4

Question 7

a. Full Inverted Indexes

-The basic inverted index is NOT suitable for answering **phrase** or **proximity queries**, because it does not contain information about where exactly the document in each word occurs

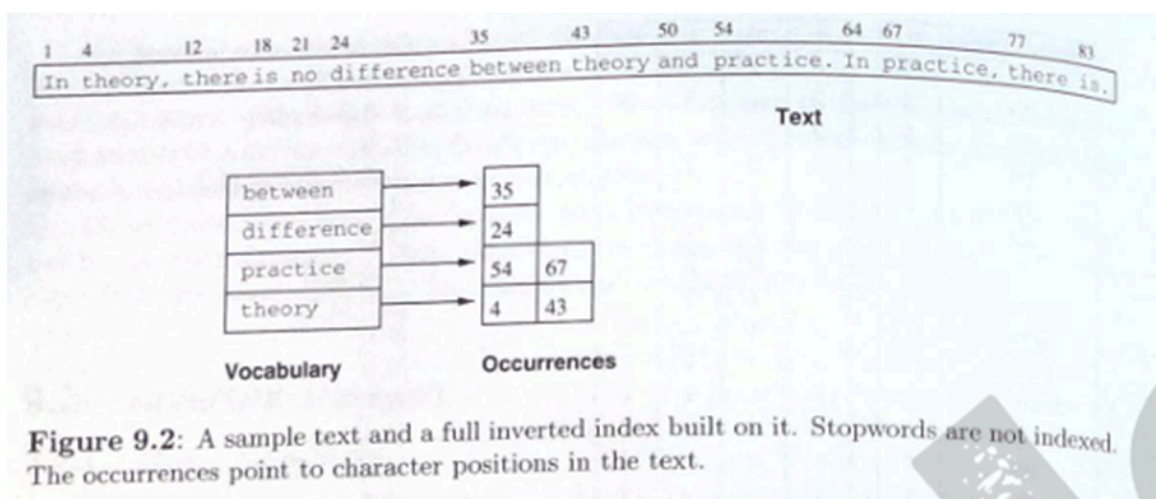
-To do this we need to add the positions of each word in each document to the index.

Word positions simplify phrase and proximity queries, while

Character positions facilitate direct access to the matching text positions.

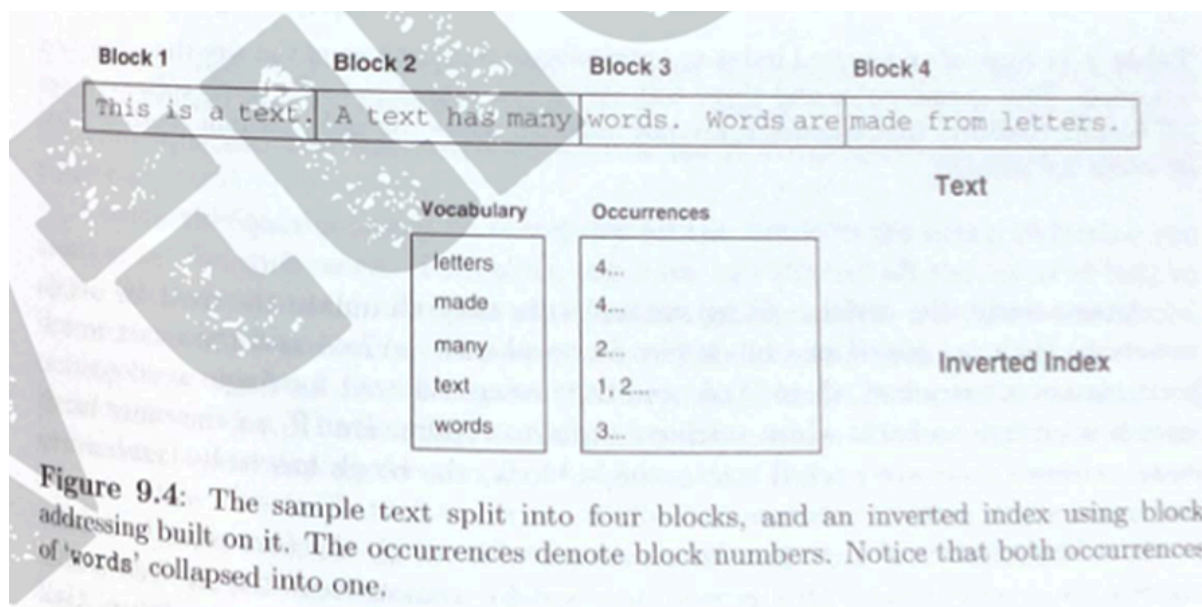
-This type of inverted index is called a *full inverted index*.

-The space required to store a full inverted index will be proportional to the overall number of occurrences, which is in turn proportional to the size of the text.



To reduce space requirements, a technique called *block addressing* is used. The text of all the documents is divided in blocks, and the occurrences point to the blocks where the word appears (instead of the exact positions). Notice that the simple inverted index is equivalent to use documents as blocks and in general to simplify the management of blocks, the boundaries of them should coincide with logical and/or physical boundaries (that is, documents or files).

Block addressing allows the pointers to be smaller, because there are fewer blocks than positions. Also, all the occurrences of a word inside a single block are collapsed to one reference (see Figure 9.4). Indexes of only 5% overhead over the text size are obtained with this technique. The price to pay is that, if the exact occurrence positions are required (for instance, for a proximity query), then an online search over the qualifying blocks has to be performed. For instance, block addressing indexes with 256 blocks stop working well with texts of a few hundreds megabytes.



b. i) Signature Files

- Signature files are word-oriented index structures **based on hashing**.
- They pose a low overhead (10% to 20% over the text size), at the cost of forcing a sequential search over the index.
- Although the search complexity is linear (as opposed to sublinear), its **constant is rather low**, which makes the technique suitable for not very large texts.
- While inverted indexes outperform signatures files for most applications

Structure

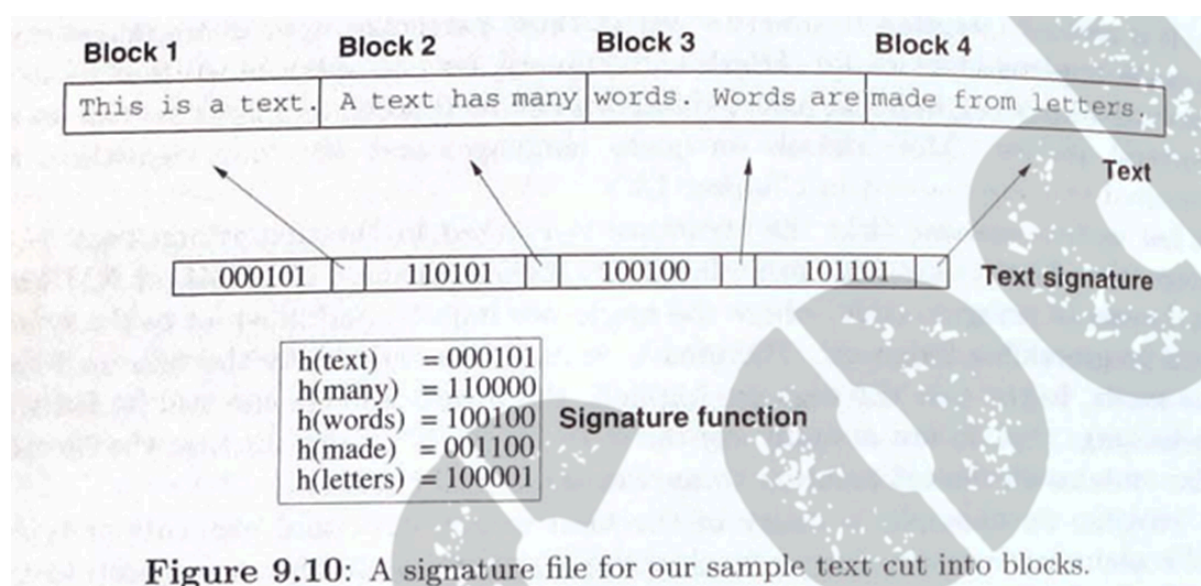
- A signature file uses a hash function(signature) that maps word blocks to bitmasks of size **B** bits.

- It divides the text in blocks of **b** words each, and to each block assigns a bit mask of size B.
- The bitmask is obtained by **bit-wise ORing** the signatures of all the words in the text block.

-Hence, the signature file is no more than a sequence and bitmasks of all blocks (plus a pointer to each block)

-The main idea is that if a word is **present** in a text block, then all the bits set in its signature are also set in the bitmask of the text block.

Hence, whenever a bit is set in the mask of the query word and not in the mask of the text block, then the word is not present in the text block



Searching

-Searching a single word is carried out by hashing it to a bitmask W , and then comparing the bitmasks B_i of all the text blocks.

-**Whenever $W \& B_i = W$** the text block *may* contain the word.

Hence for all candidate text blocks, an online traversal must be performed to verify if the word is actually there.

avoided as in inverted indexes (except if the risk of a false drop is accepted). No other types of patterns can be searched in this scheme. On the other hand, the scheme is more efficient to search phrases and reasonable proximity queries. This is because *all* the words must be present in a block in order for that block to hold the phrase or the proximity query. Hence, the bit-wise OR of all the query masks is searched, so that *all* their bits must be present. This reduces the probability of false

Construction

-The construction of a signature file is rather easy.

The text is simply cut in blocks and for each block an **entry** of the signature file is generated.

-This **entry** is the bit-wise OR of the signatures of all the words in the block

-**Adding text** can be done easily, since it is only necessary to keep **adding records** to the signature file.

-**Text deletion** is carried out by **deleting the appropriate bitmasks**.

Other storage proposals exist apart from storing all the bit masks in sequence. For instance, it is possible to make a different file for each bit of the mask, i.e. one file holding all the first bits, another file for all the second bits, etc. This reduces the disk times to search for a query, since only the files corresponding to the l bits which are set in the query have to be traversed.

Compression

-There are many alternative ways to compress signature files.

All of them are based on the fact that **only a few bits are set** in the **whole file**

-It is then possible to use efficient methods to code the bits which are not set, for **instance run-length encoding**.

Different methods to code the bits which are not set, for instance run-length encoding. Different considerations arise if the file is stored as a sequence of bit masks or with one file per bit of the mask. They allow us to reduce space and hence disk times, or alternatively to increase B (so as to reduce the false drop probability) keeping the same space overhead. Compression ratios near 70% are reported.

b. ii) Structure: Tries and Suffix Trees

Tries

-**Tries** or **digital search trees**, are multi-way trees that store sets of strings and are able to retrieve any string in time proportional to its length.

-A *suffix trie* is in essence a trie data structure built over all the suffixes of the text $T = t_1 t_2 \dots t_n$, $t_n = '$'$. The pointers to the suffixes $t_i \dots t_n$ are stored at the final states, i.e the leaves.

-To reduce the **number of nodes in a trie**, the suffix trie **removes all unary paths** that finish at a leaf.

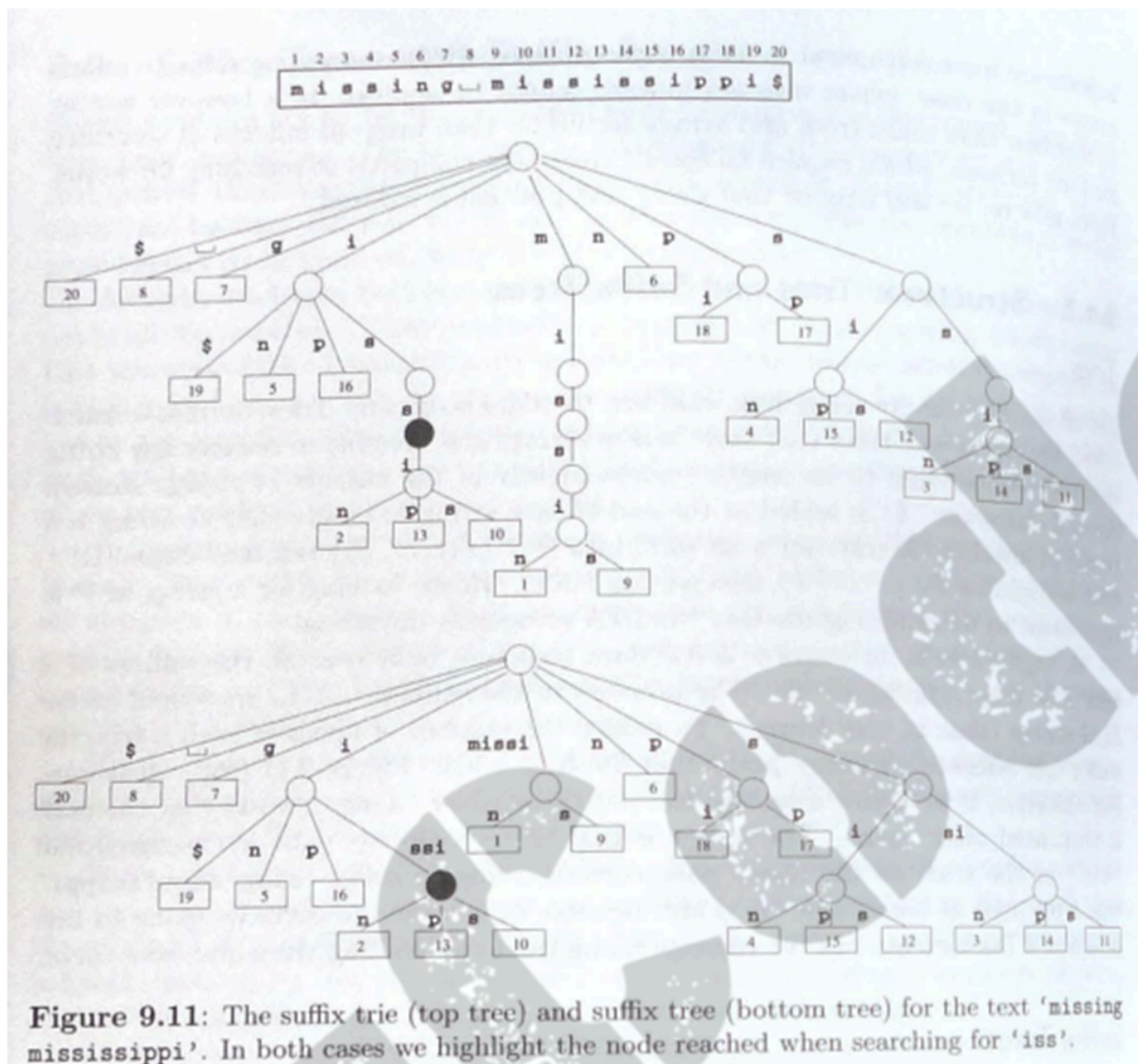


Figure 9.11: The suffix trie (top tree) and suffix tree (bottom tree) for the text 'missing mississippi'. In both cases we highlight the node reached when searching for 'iss'.

Suffix Trees

-To further reduce the space requirement, all the **remaining unary paths can be compressed**. The result is a *suffix tree*

-Since there are **n leaves** and every **internal node has at least two children**, the total size of the suffix tree is $O(n)$.

-Edges in a suffix tree are labelled by strings in general.

-The problem with suffix trees are the space requirement. Depending on the implementation the suffix tree takes 10 to 20 times the space of the text itself.

-The main property that permits finding all text substring equal to a given pattern string $P = p_1p_2 \dots p_m$.

Every text substring is a prefix of a text suffix.

-The main idea is to **descend into a trie** by following all the characters of P.

>This drives us to the last node shared by all the trie suffixes that start with P.

>The leaves that descend from the node are the starting positions of the suffixes that start with P. i.e **they point precisely to the occurrences of P in T**.

-A **suffix tree** for a text of n characters **can be built in $O(n)$ time**. The algorithm however, performs poorly in practice if the suffix tree does not fit in main memory.

-This is especially stringent, because of the large space requirements of the suffix trees.

Question 8

a. Suffix trees and Arrays

-**Suffix trees** and **Suffix arrays** enable indexed searching for any text substring matching a query string or a complex pattern

-These indexes regard the text as one long string. Each position in the text is considered as a text *suffix*(that is a string that goes from that text position to the end of the text.)

-Suffix trees and arrays are **better** than inverted indexes at **searching for long phrases**, which require no special treatment compared to searching for words.

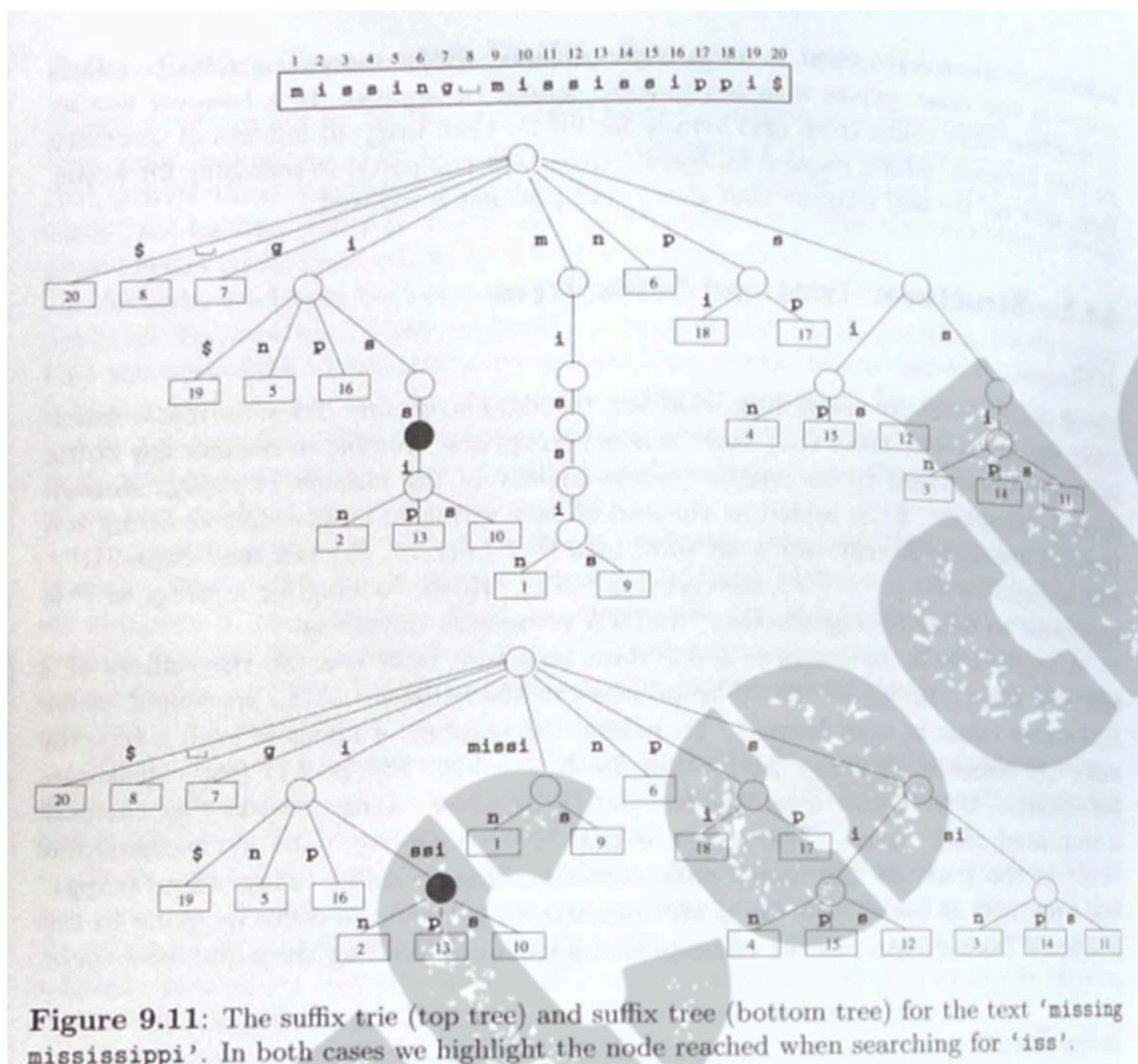
Structure: Tries and Suffix Trees

Tries

-**Tries** or **digital search trees**, are multi-way trees that store sets of strings and are able to retrieve any string in time proportional to its length.

-A *suffix trie* is in essence a trie data structure built over all the suffixes of the text $T = t_1t_2\dots t_n$, $t_n = '$'$. The pointers to the suffixes $t_i\dots t_n$ are stored at the final states. i.e the leaves.

-To reduce the **number of nodes in a trie**, the suffix trie **removes all unary paths** that finish at a leaf.



Suffix Trees

- To further reduce the space requirement, all the **remaining unary paths can be compressed**. The result is a *suffix tree*
- Since there are **n leaves** and every **internal node has at least two children**, the total size of the suffix tree is $O(n)$.
- Edges in a suffix tree are labelled by strings in general.
- The problem with suffix trees are the space requirement. Depending on the implementation the suffix tree takes 10 to 20 times the space of the text itself.

Suffix Arrays

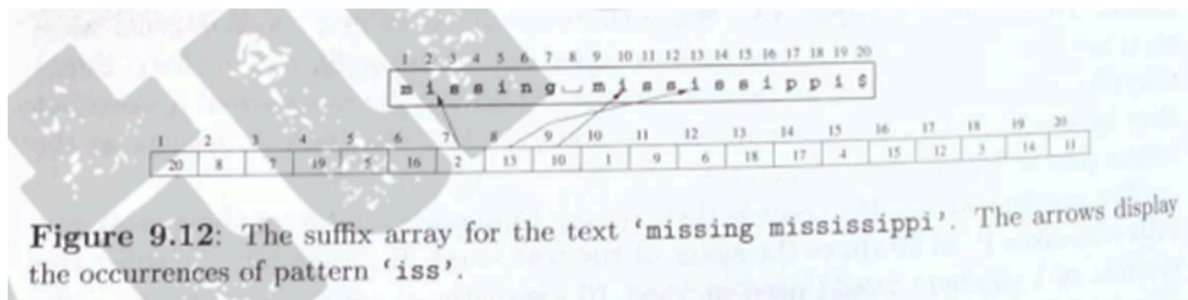
- Suffix Arrays** provide essentially the same functionality as suffix trees with much lower space requirements.

-If the children of the suffix tree nodes are **sorted left-to-right** in lexicographical edge-label order

-The suffix array is obtained by collecting all the tree leaves in **left-to-right order**.

-A suffix array of T is defined as the array **pointing to all the suffixes of T** where the suffixes **have been lexicographically sorted**.

-A suffix array takes **typically four(4) times the text size** which makes it appealing for longer texts



Searching

-The main property that permits finding all text substring equal to a given pattern string $P = p_1p_2 \dots p_m$.

Every text substring is a prefix of a text suffix.

-The main idea is to **descend into a trie** by following all the characters of P .

>This drives us to the last node shared by all the trie suffixes that start with P .

>The leaves that descend from the node are the starting positions of the suffixes that start with P . i.e **they point precisely to the occurrences of P in T** .

-Searching in a **Suffix Array** is slightly different,

>Since all the suffixes sharing prefix P are lexicographically contiguous, we can find the suffix array interval containing all the answers via **two binary searches**, that find the first and the last suffixes of P .

>Since each step in the binary search requires comparing P against a text suffix, the search cost is $O(m \log n)$.

Searching for Complex Patterns

-Searching for a complex pattern using a suffix trie is accomplished by simulating the corresponding **sequential algorithm** and **backtracking** over the trie.

-If we wish to search for a certain **regular expression**. We build it's automaton, to detect all the **text suffixes** that start with a string matching the **regular expression**

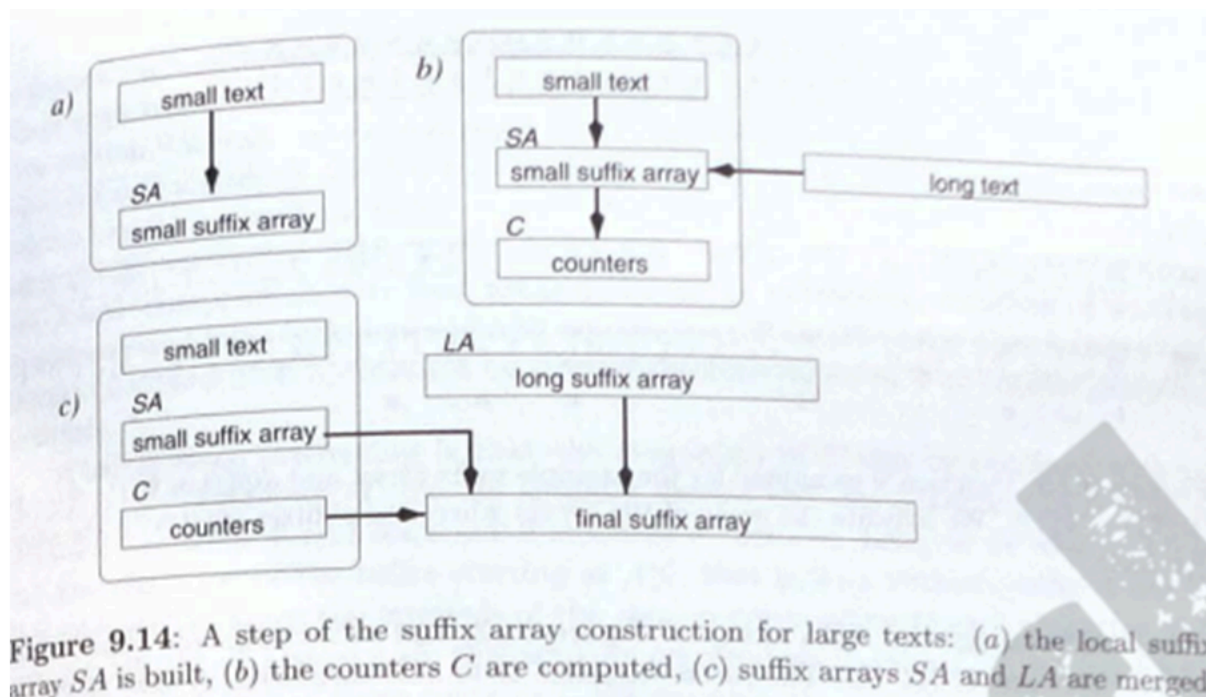
- The algorithm begins at the trie root,
For each child of the current node, (labelled by character c),
>The **automaton** is fed with c and the **algorithm enters the subtree**.
>When the recursion returns from the subtree, the **original automaton state** before feeding it with c is **restored**.
>The process is repeated for each children of the current node

Construction

- A **suffix tree** for a text of n characters **can be built in $O(n)$ time**. The algorithm however, performs poorly in practice if the suffix tree does not fit in main memory.
- This is especially stringent, because of the large space requirements of the suffix trees.
- We concentrate more on **direct suffix array construction**,
Since the **suffix array** is no more than a set of text pointers in lexicographic order, a simple way to generate it is to lexicographically sort all the pointed suffixes using any classic sorting algorithm.

Construction of Suffix Arrays for Large Texts

- When the data does not fit into the main memory. A specific algorithm for secondary memory construction is required.



Compressed Suffix Arrays

- Compressed suffix arrays *replace* the text, in the sense that they are able to **reproduce any text substring** and therefore, the text needs not be stored.
These are called self-indexes.

-We present the two main approaches to suffix array compression.

a) Using the Function ψ

b) Using the Burrows-Wheeler Transform

b. i) Faster Bit- Parallel Algorithms

-There are some bit-parallel algorithms that can handle complex patterns and still **skip text-characters**

-The algorithm runs progressively slower as the pattern gets more complex

Suffix Automata

The *suffix automaton* of a pattern P is an automaton that recognizes all the suffixes of P .

-The **BNDM algorithm** is an example of the implementation of the Suffix Automaton using bit-parallelism and achieves improved performance when the pattern is not very long.

of P . The non-deterministic version of this automaton has a very regular structure, as Figure 9.25 illustrates.



Figure 9.25: A non-deterministic suffix automaton for $P = \text{'abracadabra'}$.

BDM Algorithm

(which is faster on natural language text). To search for pattern P , the automaton of P^{rev} (the reversed pattern) is built. The algorithm scans the text window backwards (right to left) and feeds the characters into the suffix automaton of P^{rev} . If the automaton runs out of active states after scanning $t_{i+m}t_{i+m-1} \dots t_{i+j}$, this means that $t_{i+j}t_{i+j+1} \dots t_{i+m}$ is not a substring of P . Therefore, no occurrence of P can contain this text substring, and thus the window can be safely shifted past t_{i+j} . If, instead, we reach the beginning of the window and the automaton still has active states, this means that the window is equal to the pattern (check Figure 9.25), so we report the occurrence and shift the window by 1.

BNDM ($T = t_1 t_2 \dots t_n, P = p_1 p_2 \dots p_m$)

```

(1) for  $c \in \Sigma$  do  $B[c] \leftarrow 0$ 
(2) for  $j \leftarrow 1 \dots m$  do  $B[p_j] \leftarrow B[p_j] \mid (1 \ll (m - j))$ 
(3)  $i \leftarrow 0$ 
(4) while  $i \leq n - m$  do {
(5)    $j \leftarrow m - 1$ 
(6)    $D \leftarrow B[t_{i+m}]$ 
(7)   while  $j > 0 \wedge D \neq 0$  do
(8)      $D \leftarrow (D \ll 1) \& B[t_{i+j}]$ 
(9)      $j \leftarrow j - 1$ 
(10)  }
(11)  if  $D \neq 0$  then report an occurrence at text position  $i + 1$ 
(12)   $i \leftarrow i + j + 1$ 
(13) }
```

Figure 9.26: Pseudocode for BNDM algorithm.

Interlaced Shift-AND

-Another idea to achieve optimal search time is to read one text character out of q

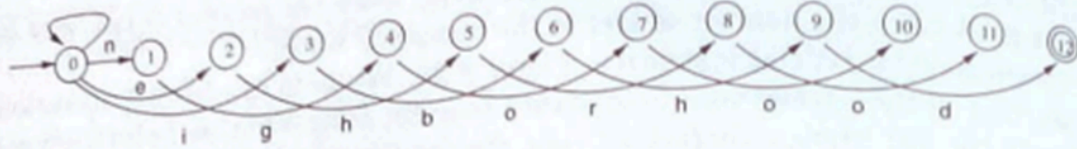


Figure 9.27: A non-deterministic suffix automaton for interlaced searching of $P =$ 'neighborhood' with $q = 3$.

Interlaced-Shift-And ($T = t_1 t_2 \dots t_n$, $P = p_1 p_2 \dots p_m$, q)

```

(1) for  $c \in \Sigma$  do  $B[c] \leftarrow 0$ 
(2) for  $j \leftarrow 1 \dots m$  do  $B[p_j] \leftarrow B[p_j] \mid (1 \ll (j - 1))$ 
(3)  $S \leftarrow (1 \ll q) - 1$ 
(4)  $D \leftarrow 0$ 
(5) for  $i \leftarrow 1 \dots \lfloor n/q \rfloor$  do {
(6)    $D \leftarrow ((D \ll q) \mid S) \& B[t_{q \cdot i}]$ 
(7)   if  $D \& (S \ll (\lfloor m/q \rfloor \cdot q - q)) \neq 0$ 
(8)   then run Shift-And over  $t_{q \cdot i - m + 1} \dots t_{q \cdot i + q - 1}$ 
}
```

Figure 9.28: Pseudocode for Interlaced Shift-And algorithm with sampling step q (simplified). It is assumed that $m \geq q$.

It is not hard to show that, if q is chosen as $m/(2 \log_\sigma m)$, then the average search time of the algorithm is the optimal $O(n \log_\sigma(m)/m)$ (assuming $m \leq w$). Yet, it is not clear that the more complex patterns of Figures 9.22 and 9.23 can be handled with this technique.

b. ii) Multi-dimensional indexing

-In multimedia data, we can represent every object by several numerical features.
 -One way to search in this case is to map these object features into points on a **multi-dimensional space** and to use **multi-attribute access** methods. (referred to as *spatial access methods* or SAMs) to cluster them and to search for them

-Another way to approach is to have a distance function for objects and then use a distance-based index(called *metric access methods* or MAMs)

-The main mapping methods form three main classes:

- (1) **R*trees** and the rest of the R-tree family
- (2) **Linear Quadtrees**
- (3) **Grid-Files**

The main methods are (1) the R-tree family, (2) linear quadtrees, and (3) grid files. Several of these methods explode exponentially with the dimensionality, eventually reducing to sequential scanning. For linear quadtrees, the effort is proportional to the hyper-surface of the query region [545]; the hyper-surface grows exponentially with the dimensionality. Grid files face similar problems, since they require a directory that grows exponentially with the dimensionality. The R-tree-based methods seem to be most robust for higher dimensions, provided that the fan-out of the R-tree nodes remains > 2 . Below, we give a brief description of the R-tree method and its variants, since it is one of the typical representatives of spatial access methods.

The R-tree represents a spatial object by its minimum bounding rectangle (MBR). Data rectangles are grouped to form parent nodes, which are recursively grouped to form grandparent nodes and, eventually, a tree hierarchy. The MBR of a parent node completely contains the MBRs of its children; MBRs are allowed to overlap. Nodes of the tree correspond to disk pages. Disk pages, or 'disk blocks', are consecutive byte positions on the surface of the disk that are typically fetched with one disk access. The goal of the insertion, split, and deletion routines is to give trees that will have good clustering, with few, tight parent MBRs. Figure 9.36 illustrates data rectangles (in black), organized in an R-tree with fan-out 3. Figure 9.37 shows the file structure for the same R-tree, where nodes correspond to disk pages.

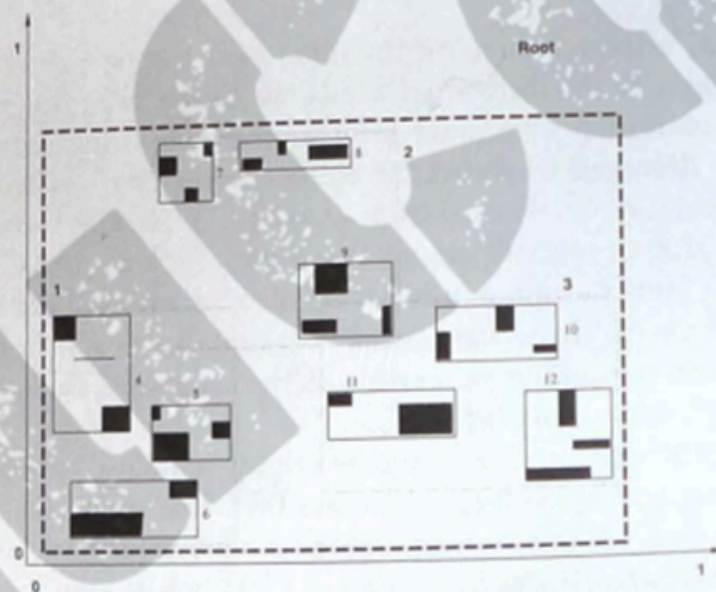
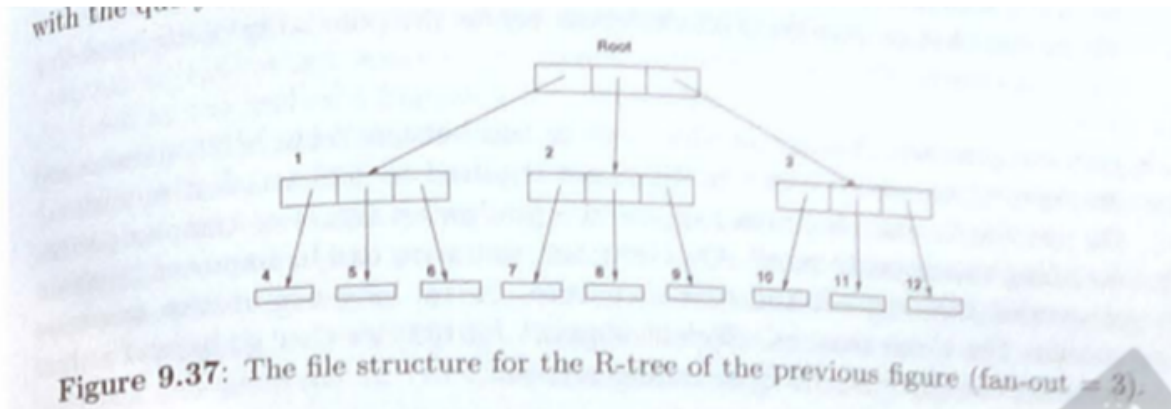


Figure 9.36: Data (dark rectangles) organized in an R-tree with fan-out = 3. Solid, light-dashed, and heavy-dashed lines indicate parents, grandparents and great-grandparent (the root, in this example).



-The R*-tree is structurally identical to an R-tree; the main difference is a clever **improvement on the split algorithm**, based on the concept of **forced reinsert**.

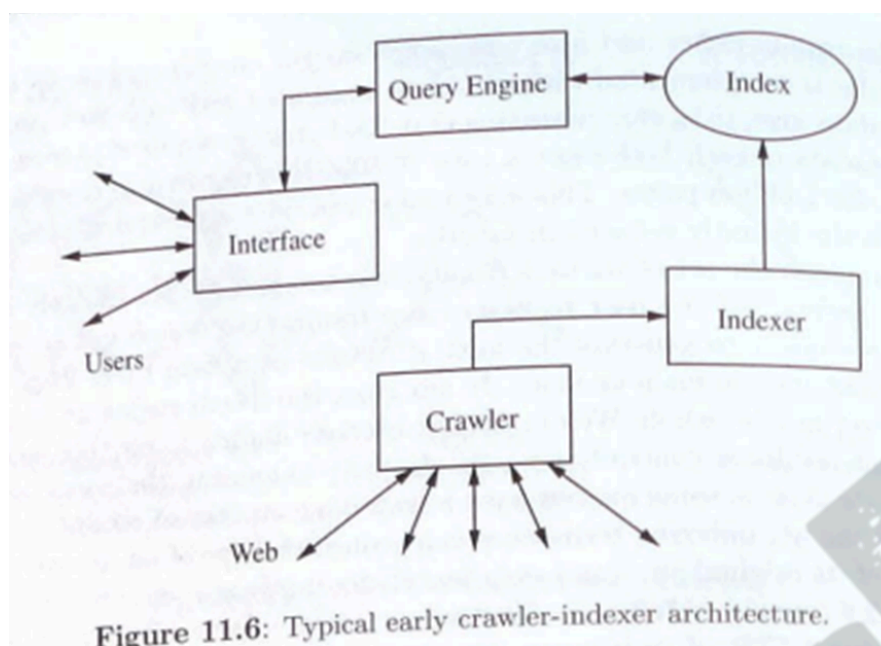
When a node overflows, some of its children are carefully chosen; they are deleted and reinserted, usually resulting in a R-tree with a better structure

Module 5

Question 9

a. Search Engine Architecture

-Most search engines use a *centralized crawler-indexer architecture*.



- **Crawlers**(also called Spiders, Robots, Wanderers or Walkers) are Programs that traverse the Web sending new or updated pages to a main server.
Crawler runs on a local system and sends requests to remote Web servers.
- The **Index** is used in centralized fashion to answer queries submitted from different places on the Web.
Most search engines use variants of the **Inverted index**.
In simple terms and inverted index consists of a list of terms where each term is associated with a list of pointers to the pages in which it occurs.
- Given a **Query**, the set of answers displayed is a subset of the complete result set.
- Search engines typically only retrieve a subset of the most relevant results for a query (usually 10). They don't calculate the entire result set to save time and resources.
- Further results are computed on demand only when the user requests them, improving efficiency.
- The inverted index is a core data structure used for fast searching. It maps terms to the documents containing them. State-of-the-art techniques can significantly reduce the size of the inverted index.
- The sheer volume of the web poses a significant challenge for search engines, as they need to handle massive amounts of data. The early crawler-indexer architecture faced limitations in scaling to handle the rapidly growing web.
- To address the scalability issues, modern search engines distribute and parallelize the processing of data across multiple machines.
- The main problem faced by this architecture is the **gathering of data** and the **sheer volume of data**
- In fact, the crawler-indexer architecture was not able to cope with the Web growth at the end of the 1990s.
- The solution was to distribute and parallelize computation.

b. Cluster based Architecture

- Current search engines use a **massive parallel** and **cluster-based** architecture.
- Due to the large size of the document collection, the inverted index does not fit in a single computer and must be distributed across the computers of a cluster
- **Document partitioning** is used, The large volume of queries implies that the basic architecture must be replicated in order to handle the overall query load, and
- Each cluster must handle a subset of the query load
- Queries originate from all around the world, and internet latency is appreciable in many continents.
- **Cluster Replicas** are maintained in different geographical locations to decrease answer time.
- Cluster replicas also, allows search engines to be **fault-tolerant** in most typical worst-case scenarios, such as power
- There are also other many crucial details to be carefully addressed in this type of architecture

1. It is particularly important to achieve a good balance between the internal (answering queries and indexing) and external (crawling) activities of the search engine. This is achieved by assigning dedicated clusters to crawling, to document serving, to indexing, to user interaction, to query processing, and even to the generation of the result pages.
2. In addition, a good load balancing among the different clusters needs to be maintained. This is achieved by specialized servers called (quite trivially) *load balancers*.
3. Finally, since hardware breaks often, fault tolerance is handled at the software level. Queries are routed to the most adequate available cluster and CPUs and disks are routinely replaced upon failure, using inexpensive exchangeable hardware components.

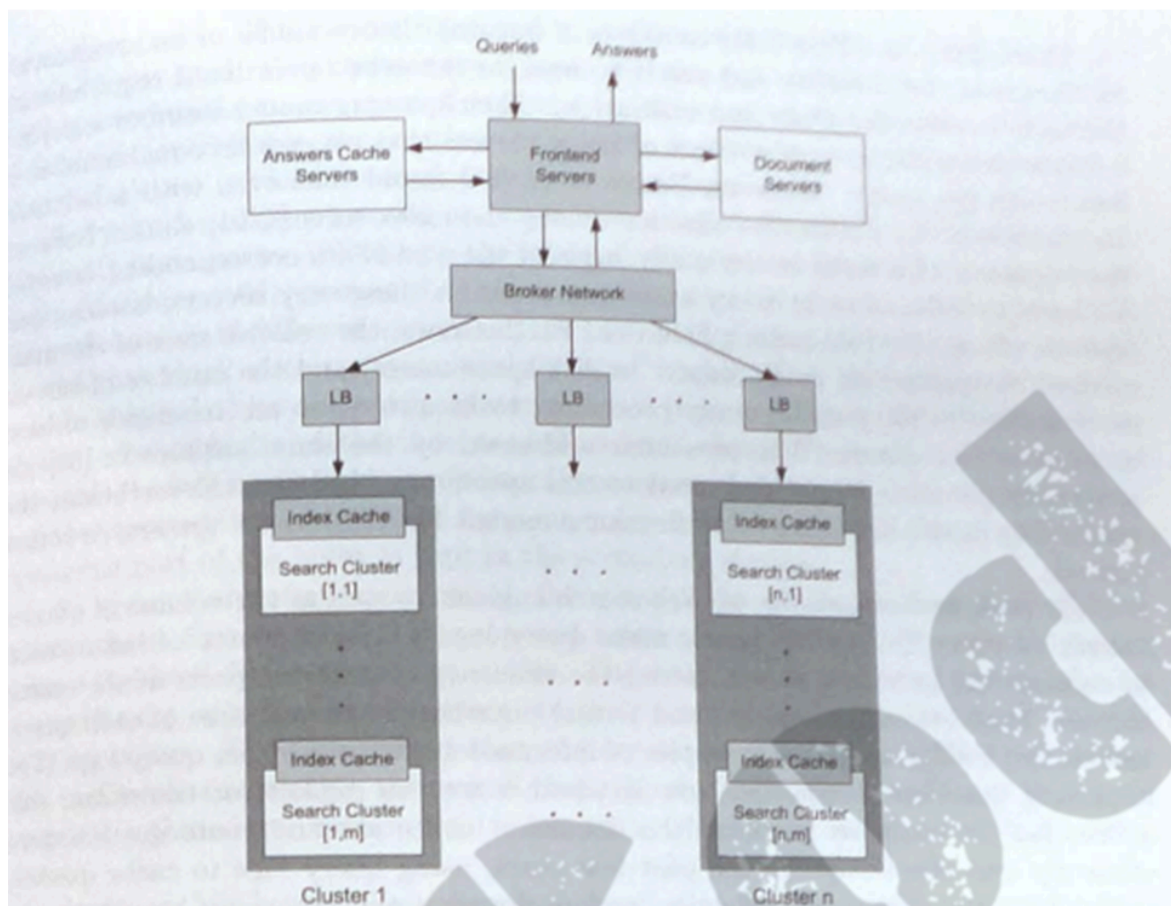


Figure 11.7: Cluster-based architecture for the search module based on [484]. Each cluster contains an index of the whole collection, i.e., the index is partitioned among the m servers in the cluster. The n clusters are used to produce n replicas of the whole index.

Question 10

a. XML Retrieval Evaluation

-Due to these particular characteristics of structured text, the evaluation of XML retrieval systems requires the **building of test collections** where the evaluation of paradigms are provided according to criteria that take into account **the inherent structural constraints** i.e the element structural relationships.

-The Initiative for Evaluation of XML retrieval (INEX) established an infrastructure in the form of large test collections and appropriate measures for evaluating XML retrieval effectiveness.

Document Collections

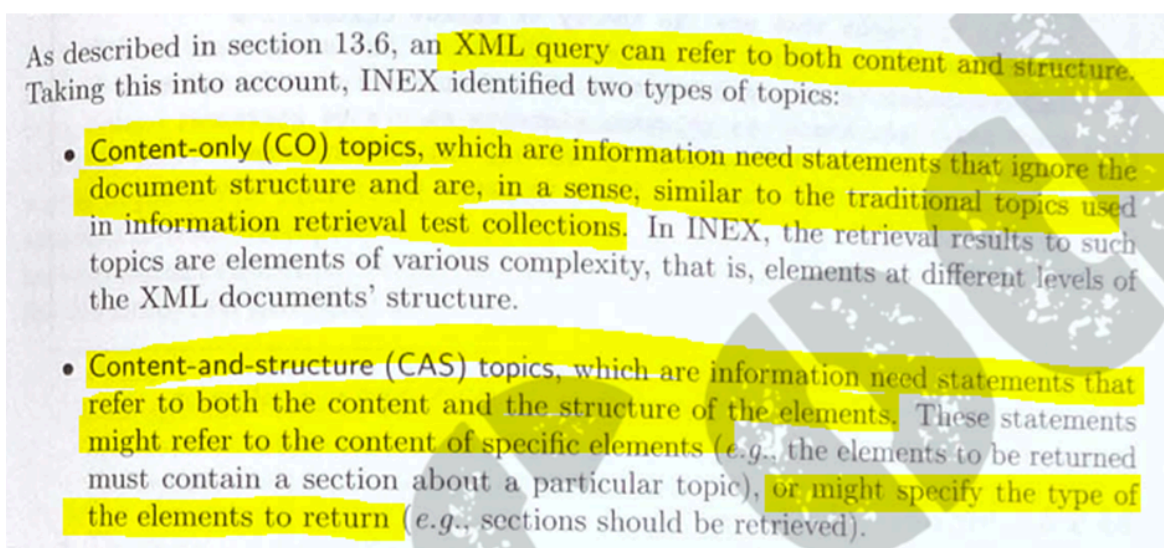
-The document collections used in INEX up to 2004 consisted of 12,107 articles marked-up in XML, from 12 magazines and 6 transactions of the IEEE Computer Society's publications.

-On Average an article contains 1532 XML nodes, where the average depth of the node is 6.9.

-In 2005, the collection was extended with further publications from the IEEE Computer Society. A total of 4712 new articles from the period of 2002-2004 were added giving a total of 16,819 articles totalling 764 MB in size and 11 million elements

-Since 2006, INEX uses a different document collection, made from English documents from Wikipedia.

Topics



As described in section 13.6, an XML query can refer to both content and structure. Taking this into account, INEX identified two types of topics:

- **Content-only (CO) topics**, which are information need statements that ignore the document structure and are, in a sense, similar to the traditional topics used in information retrieval test collections. In INEX, the retrieval results to such topics are elements of various complexity, that is, elements at different levels of the XML documents' structure.
- **Content-and-structure (CAS) topics**, which are information need statements that refer to both the content and the structure of the elements. These statements might refer to the content of specific elements (*e.g.*, the elements to be returned must contain a section about a particular topic), or might specify the type of the elements to return (*e.g.*, sections should be retrieved).

-CO and CAS topics reflect users with **varying levels of knowledge** about the structure of the collection to search for relevant information.

-CAS topics fit the needs of users who want to take **advantage of knowledge** on the **document structure** to improve the quality of results.

Retrieval Tasks

-A major departure from traditional flat document retrieval is that XML retrieval systems need not only score elements with respect to their **likelihood relevance to a query**, but Also need to determine the appropriate level of element granularity to return to it's users.

-The task of an XML retrieval system in INEX was to return **instead of whole documents**, those XML elements that are most relevant i.e most specific and exhaustive to the user's query.

-In other words XML systems should return components that contain as much relevant information and as little irrelevant information as possible

-With this generic task, the actual relationship between retrieved elements was not considered and many systems **retrieved overlapping elements**

- CO sub-task, which makes use of the CO topics, where an effective XML retrieval system is one that retrieves the most specific elements, and only those which are relevant to the topic of request.
 - CAS sub-task, which makes use of CAS topics, where an effective system is one that retrieves the most specific document components that are relevant to the topic of request and match, either strictly or approximately (vaguely), the structural constraints specified in the query. This led to two CAS sub-tasks:
 - SCAS sub-task (strict content-and-structure), where a strict interpretation of the structural constraints was adopted. This sub-task was investigated in 2002 and 2003.
 - VCAS sub-task (vague content-and-structure), where the goal of an XML retrieval system was to return relevant elements that may not exactly conform to the structural conditions expressed within the query, but where the path specifications should be considered hints as to where to look to find relevant content.
- This sub-task was introduced in 2003, and since 2004, only this CAS sub-task was investigated. The SCAS sub-task was felt to be an unrealistic task because specifying an information need is not an easy task, in particular for XML documents with a wide variety of tag names (e.g., <title>).

Relevance

-In terms of traditional document retrieval, it is usually understood as the connection between **a retrieved item** and **the user's query**

-In XML retrieval, the relationship between a **retrieved item** and the **user's query** is further complicated by the need to consider the structured in the documents.

-Since retrieved elements can be at any level of granularity, an element and one its **children** can both be given as relevant to a query.

The child element may be more focused on the query than its **parent element**, which may contain **additional irrelevant content**

In this case, the child element is better to retrieve than the parent because it is more **SPECIFIC** to the query.

-In addition a multiple degree scale was necessary to allow explicit representation of how **exhaustively a topic is discussed** within an element with **respect to its children elements**.

Measures

-Measuring XML Retrieval effectiveness requires **considering the dependencies** among the elements.

-Users in XML retrieval have access to other **structurally related elements** in the result set. They may hence locate **additional relevant information** by browsing or scrolling down the element.

Hence this motivates the need to consider so called **near-misses**, which are elements from where users can access relevant content

-More formally let **hlength(e)** be the length in characters of highlighted text contained in element **e** for a given topic

-Let **length(e)** be the total number of characters contained in element **e**,

-And let **T_{rel}** be the total amount of (highlighted) relevant information in the collection for the topic.

-Let **erank(i)** be a function that returns the element at ranked **i**.

Precision at rank **r**, indicated by **P@r**, is the fraction of retrieved relevant information upto rank **r**

$$P@r = \frac{\sum_{i=1}^r hlength(erank(i))}{\sum_{i=1}^r length(erank(i))}$$

-The definition ensures that, to achieve a high precision at rank **r**, the elements retrieved up to and including that rank need to contain as **little non-relevant information as possible**.

-Recall at rank **r**, indicated by **R@r**, is the fraction of relevant information retrieved up to rank **r**.

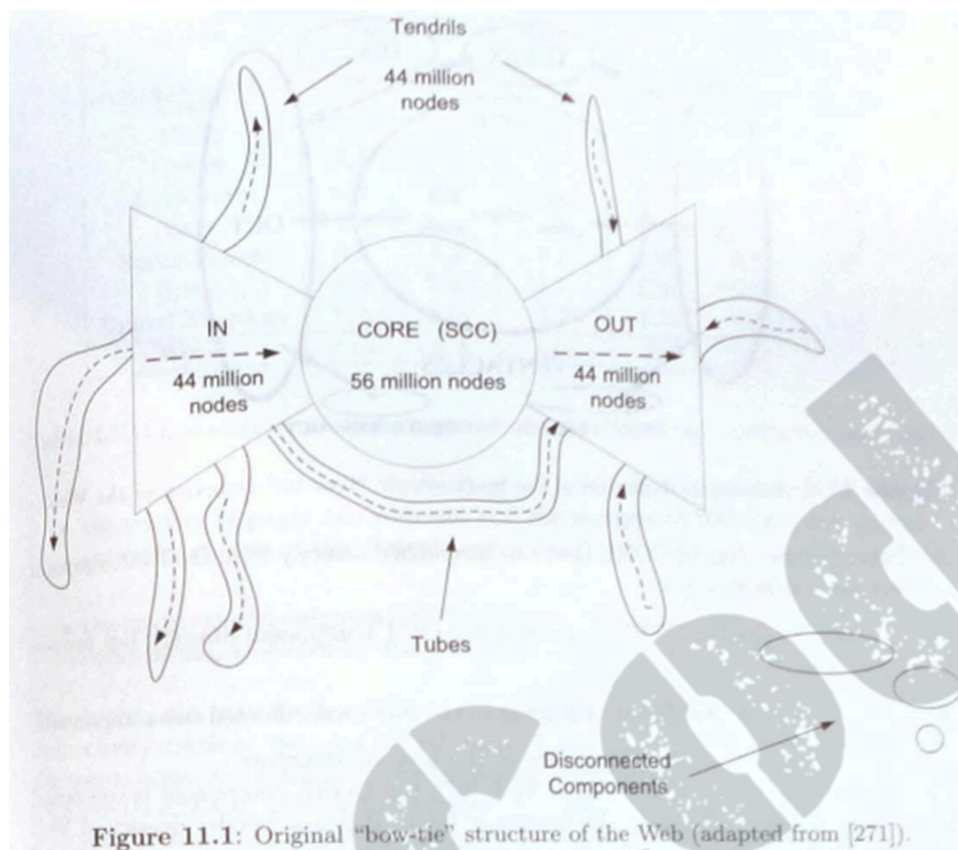
-This definition ensures that to achieve a **high recall value** at rank **r**, the elements retrieved up to and including that rank need to contain as **much relevant information as possible**.

$$R@r = \frac{1}{T_{rel}} \times \sum_{i=1}^r hlength(erank(i))$$

- T_{rel} depends on whether returning overlapped elements is allowed or not
 - >For the **thorough sub-task**, T_{rel} is the total number of highlighted characters across all elements
 - >For the **focused sub-task**, T_{rel} is the total number of highlighted characters across all documents.

b. i) Structure of Web Graph

- It is commonly agreed that the Web can be viewed as a graph in which, the **Nodes** represent individual pages and the **edges** represent links between pages
- The most complete study was conducted by Broder et al, and it compared the topology of the Web graph to a *bow-tie*.

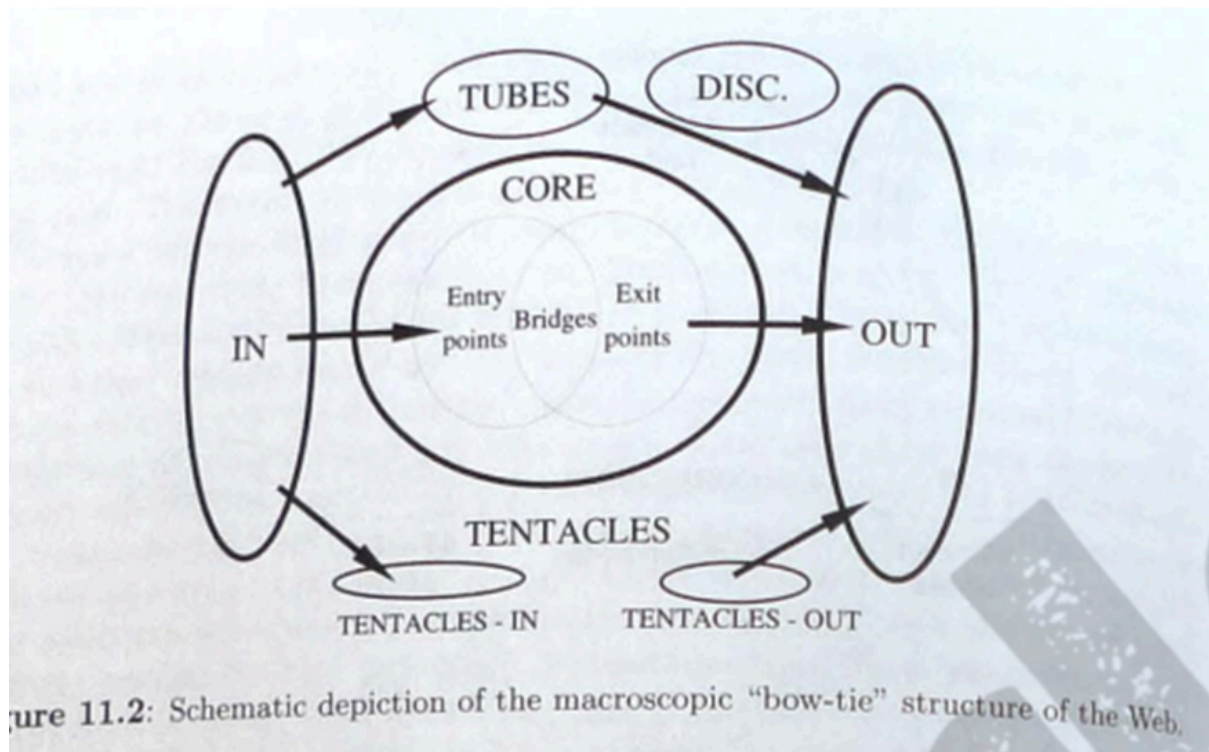


The components identified in the graph were the following:-

- a. **CORE:** sites that compose the strongly connected component of the graph. By definition one can navigate from any site in CORE to any other site in CORE.
- b. **IN:** sites that can reach sites in CORE but cannot be reached from sites in CORE
- c. **OUT:** sites that can be reached from sites in CORE, but without a path to go back to CORE.
- d. **TUBES:** Sites in paths that connect directly IN to OUT and that are outside CORE.
- e. **TENTACLES** or **TENDRILS:** Sites that can be reached from sites in **IN** and sites that only reach sites in **OUT**, but do not belong to the previous components
- f. **DISCONNECTED** or **ISLANDS:** unconnected sites whose connected component have a similar structure to the whole web

This notation was extended by dividing the CORE components into four parts as described:-

- a. **Bridges:** Sites in CORE that can be reached directly from the IN component and that can reach directly the OUT component
- b. **Entry Points:** Sites in CORE that can be reached directly from the IN component but are not bridges
- c. **Exit Points:** Sites in CORE that reach OUT component directly, but are not in Bridges
- d. **Normal:** Sites in CORE not belonging to the previously defined sub-components



b. ii) Link based Ranking

- Given that there might be thousands or even millions of pages available for any given query,
 - the problem of *ranking* those pages **to generate a short list** is probably one of the key problems in Web IR
- The **number of hyperlinks** that point to a page provides a measure of its **popularity and quality**.
 - Further, many links in **common among pages** and pages referenced by a same page are often indicative of page relations with potential value for ranking purposes.

Early Algorithms

One signal of importance for a scientific paper is the number of citations that the article receives, a topic that researchers in library sciences have studied for long [880]. Based on this idea, several authors proposed to use incoming links for ranking Web pages [1086, 855, 1021]. However, it quickly became clear that just counting links was not a very reliable measure of authoritativeness (it was not in scientific citations either), because it is very easy to externally influence this count by creating new pages (which costs nearly nothing).

Yuwono and Lee [1760] proposed three ranking algorithms in addition to the classic TF-IDF scheme (see Chapter 3), namely: Boolean spread, vector spread, and most-cited. The first two are the normal ranking algorithms of the Boolean and vector space models extended to include pages pointed by a page in the answer or pages that point to a page in the answer. The third algorithm, "most-cited", is based only on the terms included in pages having a link to the pages in the answer. Their comparative study of these techniques considering 56 queries over a collection of 2,400 Web pages indicated that the vector spread model yields a better recall-precision curve, with an average precision of 75%.

Another early example is WebQuery [338], which also allows visual browsing of Web pages. WebQuery takes a set of Web pages (for example, a list of search results) and ranks them based on how connected each Web page is. Additionally, it extends the set by finding Web pages that are highly connected to the original set. A related approach is presented by Li [1021].

HITS

-This ranking scheme is query-dependent, and considers the set of pages **S** that **point to** or **are pointed by** pages in the answer.

-Pages that have many links pointing to it in **S** are called **authorities**
Because they are susceptible to contain authoritative and relevant content

-Pages that have many outgoing links are called **hubs**
Because they are susceptible to point to relevant similar content.

A positive two-way feedback exists:

- >Better **authority pages** come from incoming edges from **good hubs** and
- >better **hub pages** come from outgoing edges to **good authorities**.

come from outgoing edges to good authorities. Let $H(p)$ and $A(p)$ be the hub and authority values of page p . These values are defined such that the following equations are satisfied for all pages p :

$$H(p) = \sum_{u \in S \mid p \rightarrow u} A(u), \quad A(p) = \sum_{v \in S \mid v \rightarrow p} H(v) \quad (11.5)$$

where $H(p)$ and $A(p)$ for all pages are normalized (in the original paper, the sum of the values of each measure is not fixed).

PageRank

-The best known link-based weight is PageRank, which is part of the ranking algorithm originally used by **Google**

-PageRank simulates **a user navigating randomly on the Web**.

-After a **large number of moves** we can **compute the probability** with which our user visited each page.

This probability is a property of the graph, which was referred to as PageRank in the context of the Web.

-An additional case is considered in which a the "user" can jump to any other page in the graph with a small probability q .

This is done to avoid **dead ends** i.e links without outgoing links and also **self-links**

to any other page in the graph, with a small probability q . Hence, our user jumps to a random page of the Web graph with probability q or follows one of the hyperlinks in the current page with probability $1 - q$. By the definition of this random walk on the Web graph, this user never goes back to a page just visited following an already traversed hyperlink backwards. This process can be modeled by a Markov chain, from which the stationary probability of being in each page can be computed. Let $L(p)$ be the number of outgoing links of page p and suppose that page a is pointed by pages p_1 to p_n . Then, the PageRank of a page a is given by the probability $PR(a)$ of finding our user in that page and is defined by

$$PR(a) = \frac{q}{T} + (1 - q) \sum_{i=1}^n \frac{PR(p_i)}{L(p_i)} \quad (11.6)$$