Unit 3 TEXT CLASSIFICATION AND CLUSTERING

A Characterization of Text Classification – Unsupervised Algorithms: Clustering – Naïve Text Classification – Supervised Algorithms – Decision Tree – k-NN Classifier – SVM Classifier – Feature Selection or Dimensionality Reduction – Evaluation metrics – Accuracy and Error – Organizing the classes – Indexing and Searching – Inverted Indexes – Sequential Searching – Multi-dimensional Indexing.

3.1 Web search overview

The World Wide Web allows people to share information globally. The amount of information grows without bound. In order to extract information the user needs a tool to search the Web. The tool is called a **search engine.**

3.1.1 Search Engines

A Web search engine is a software system that is designed to search for information on the World Wide Web. It uses the keywords to search for documents that relate to these key words and then puts the result in order of relevance to the topic that was searched for.

Examples of search engines

3.1.2 Types of search engines

1.

Crawler-based search engines use automated software programs to survey and categorize web pages. The programs used by the search engines to access our web pages are called _spiders‘, _crawlers‘, _robots‘ or _bots‘.

Examples of crawler-based search engines are:

a)Google (www.google.com) b) Ask Jeeves (www.ask.com)

2. Directories

A _directory‘ uses human editors who decide what category the site belongs to, they place websites within specific categories in the _directories‘ database. The human editors comprehensively check the website and rank it, based on the information they find, using a pre-defined set of rules.

There are two major directories a) Yahoo Directory (www.yahoo.com) b) Open Directory (www.dmoz.org)

3. Hybrid Search Engines

Hybrid search engines use a combination of both crawler-based results and directory results. More and more search engines these days are moving to a hybrid-based model. Examples of hybrid search engines are:

a)Yahoo (www.yahoo.com) b)Google (www.google.com)

4. Meta Search Engines

Meta search engines take the results from all the other search engines results, and combine them into one large listing. Examples of Meta search engines include: a)Meta crawler (www.metacrawler.com) b) Dogpile (www.dogpile.com)

3.1.3 Characteristics of search engines

- Unedited – anyone can enter
- Quality issues
- Spam
- Varied information types Phone book, brochures, catalogs, dissertations, news reports, weather, all in one place
- Different kinds of users Online catalog, scholars searching scholarly literature, Web
- Every type of person with every type of goal scale Hundreds of millions of searches/day,billions of docs

Differentiate between directories vs search engines

Directories	Search Engines
Hand-selected sites	All pages in all sites
Search over the contents of the descriptions of the pages	Search over the contents of the pages themselves
Organized in advance into categories	Organized after the query by relevance rankings or other scores

Types of Search Engines

- Search by Keywords (e.g. AltaVista, Excite, Google, and Northern Light)
- Search by categories (e.g. Yahoo!)
- Specialize in other languages (e.g. Chinese Yahoo! and Yahoo! Japan)
- Interview simulation (e.g. Ask Jeeves!)

3.2 Web structure

Almost every website on the Internet has a distinct design & organization structure. Website designers usually create distinguishable layout templates for pages of different functions. They then organize the website by linking various pages with hyperlinks, each of which is represented by a URL string following some pre-defined syntactic patterns.

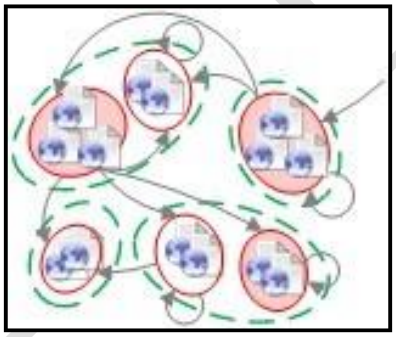


Fig. 3.2 Linkage of pages

3.2.1 Bow-Tie Structure of the Web

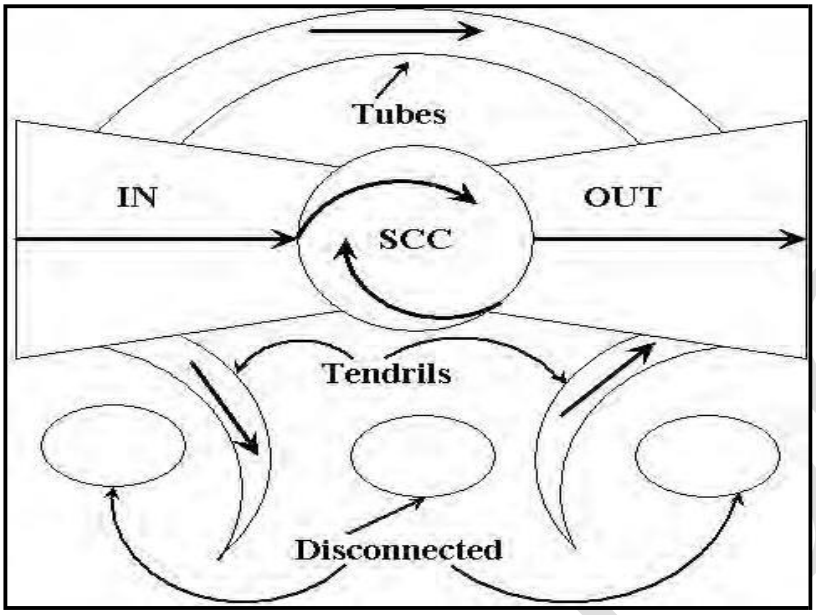


Fig.3.2.1 Bow-tie structure of Web

A web crawl is a task performed by special purpose software that surfs the Web, starting from a multitude of web pages and then continuously following the hyper links it encountered until the end of the crawl. One of the intriguing findings of this crawl was that the Web has a bow-tie structure as shown in the above figure.

The central core of the Web (the knot of the bow-tie) is the strongly connected component (SCC), which means that for any two pages in the SCC, a user can navigate from one of them to the other and back by clicking on links embedded in the pages encountered. The relative size of the SCC turned out to be 27.5% of the crawled portion of the Web. The left bow, called IN, contains pages that have a directed path of links leading to the SCC and its relative size was 21.5% of the crawled pages. Pages in the left bow might be either new pages that have not yet been linked to, or older web pages that have not become popular enough to become part of the SCC. The right bow, called OUT, contains pages that can be reached from the SCC by following a directed path of links and its relative size was also 21.5% of the crawled pages. Pages in the right bow might be pages in e-commerce sites that have a policy not to link to other sites. The other components are the ―tendrils‖ and the ―tubes‖ that together comprised 21.5% of the crawled portion of the Web, and further ―disconnected‖ components whose total size was about 8% of the crawl. A web page in Tubes has a directed path from IN to OUT bypassing the SCC, and a page in Tendrils can either be reached from IN or leads into OUT. The pages in disconnected are not even weakly connected to the SCC; that is, even if we ignored the fact that hyperlinks only allow forward navigation, allowing them to be traversed backwards as well as forwards, we still could *not* reach reach the SCC from them.

3.2.2 Small-World Structure of the Web

The study of the Web also revealed some other interesting properties regarding the structure and navigability of the Web. It turns out that over 75% of the time there is no directed path of links from one random web page to another. When such a path exists, its average distance is 16 clicks and when an undirected path exists (i.e., one allowing backward traversal of links) its average distance is only seven clicks. Thus, the Web is a *small-world network* popularly known through the notion of ―six degrees of separation,‖ where any two random people in the world can discover that there is only a short chain of at most six acquaintances between them. Since in a small-world network the average distance between any two nodes is logarithmic in the number of pages in the network, a 10-fold increase in the size of the Web would only lead to the average distance increasing by a few clicks.

The *diameter* of a graph is the maximum shortest distance between any two nodes in the graph. It was found that the diameter of the SCC of the web graph is at least 28 but when considering the Web as a whole, the diameter is at least 500; this longest path is from the most distant node in IN to the most distant node in OUT.

3.3 User Problems

There are some problems when users use the interface of a search engine.

- The users do not exactly understand how to provide a sequence of words for the search.
- The users may get unexpected answers because he/she is not aware of the input requirement of the search engine. For example, some search engines are case sensitive.
- The users have problems understanding Boolean logic: therefore, the user cannot perform advanced searching.
- Novice users do not know how to start using a search engine.
- The users do not care about advertisements, so the search engine lacks funding.
- Around 85% of users only look at the first page of the result, so relevant answers might be skipped. In order to solve the problems above, the search engine must be easy to use and provide relevant answers to the query.

3.3.1 Searching Guidelines

Here are some guidelines helping users to search.

- Specify the words clearly, stating which words should be in the page and which words should not be in the page.
- Provide as many particular terms as possible: page title, date, and country.
- If looking for a company, institution, or organization, try to guess the URL by using the www prefix followed by the name, and then (.com, .edu, .org, .gov, or country code).
- Some search engines specialize in some areas. For example, the users can use ResearchIndex (www.researchindex.com) to search research papers.
- If the users use broad queries, try to use Web directories as starting points.
- The user should notice that anyone can publish data on the Web, so information that they get from search engines might not be accurate.

3.4 Sponsored Search and Paid Placement

An effective and profitable form of advertising for search engines, pioneered by GoTo.com (that was renamed to Overture in 2001 and acquired by Yahoo in 2003), is called *paid placement* also known as *sponsored search.*

Paid placement online advertising in which links to advertisers products appear next to keyword search results, has emerged as a predominant form of Internet advertising. Under paid placement advertising, sellers (advertisers) bid payments to a search engine to be placed on its recommended list for a keyword search. A group of advertisers who bid more than the rest are selected for placement, and their positions of placement react their order of bids, with the highest bidder placed at the top position. The rapid growth of paid placement advertising has made it one of the most important Internet institutions, and has led to enormous commercial successes for search engines.

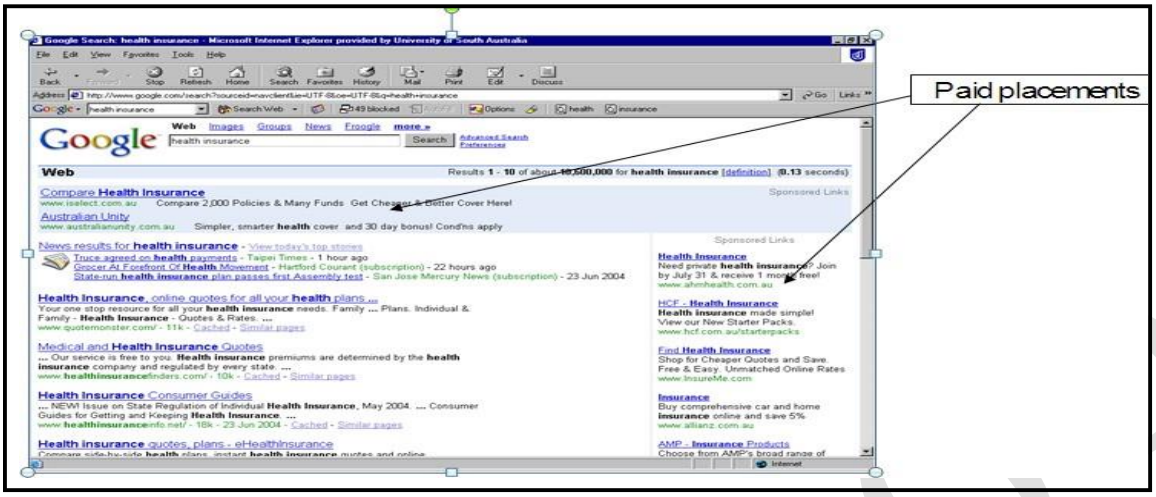


Fig 3.4.1 Example of Paid Placement

Fig 3.4.2 **Paid Placement, Regular Listings and advertising in comparison shopping engine**

In this scheme the search engine separates its query results list into two parts:
(i) an organic list, which contains the free unbiased results, displayed according to the search engine‘s ranking algorithm, and (ii) a sponsored list, which is paid for by advertising .This method of payment is called *pay per click* (PPC), also known as *cost per click* (CPC), since payment is made by the advertiser each time a user clicks on the link in the sponsored listing.

Organic Search vs Paid Search

Whenever you type a question into Google, or any other search engine, the list of links that appear below the ads are known as "organic results." These appear purely based on the quality and content of the page. Traffic that comes from people finding your links among these results is classified as "organic search" traffic or just organic traffic.

Paid search accounts are those that companies have paid to appear the top of search results

Fig 3.4.3 Example of Organic Search Results, Sponsored Links

3.5 What is SEO

Search Engine Optimization refers to set of activities that are performed to increase number of desirable visitors who come to our site via search engine. These activities may include thing we do to our site itself, such as making changes to our text and HTML code, formatting text or document to communicate directly to the search engine.

3.5.1 Define SEO

Search Engine Optimization is the process of improving the visibility of a website on organic ("natural" or un-paid) search engine result pages (SERPs), by incorporating search engine friendly elements into a website.

Search engine optimization is broken down into two basic areas: on-page, and off-page optimization. On-page optimization refers to website elements which

comprise a web page, such as HTML code, textual content, and images. Off- page optimization refers, predominantly, to back links (links pointing to the site which is being optimized, from other relevant websites).

On-Page Factors

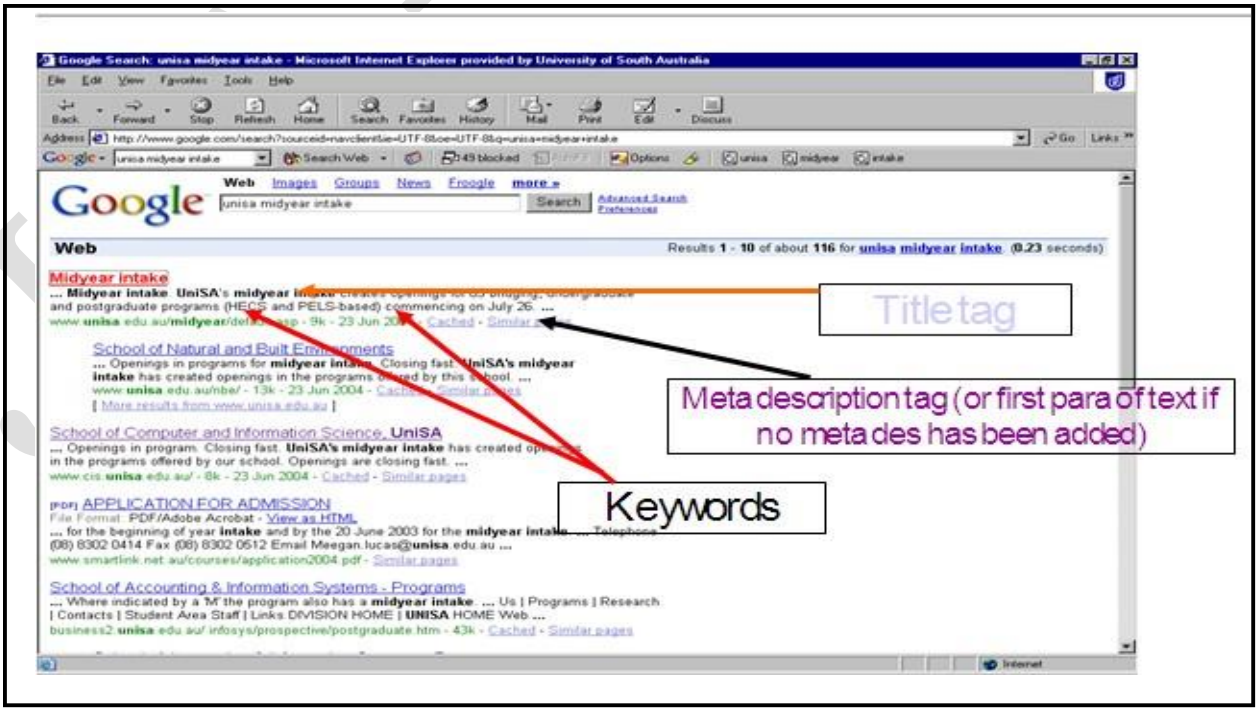
3.Title tags<title>
5.Header tags<h1>
4.ALT image tags
1. Content,Content,Content(Body text)<body>
6.Hyperlink text
2. Keyword frequency and density

Off-Page Factors

1. Anchor text
2. Link Popularity (—votes for your site) – adds credibility

3.5.2 How do organic search listings work?

- A *spider* or *crawler* which is a component of a SE gathers listings by automatically "crawling" the web
- The spider follows links to web pages, makes copies of the pages and stores them in the SE's index
- Based on this data, the SE then *indexes* the pages and *ranks* the websites
- Major SEs that index pages using spiders: Google, Altavista, Msn, Aol, lycos

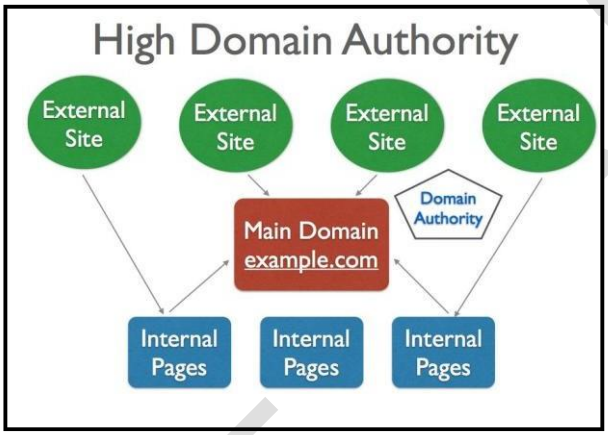


Title tag, Meta description tag and Keywords

3.5.3 The Search Engine Optimization Techniques

1

Domain naming is important to our overall foundation, use sub-directory root domains (example.com/awesome)versus sub-domains (awesome.example.com). All inbound links pointing to a website‘s main domain (www.example.com) as well as its internal pages contribute to the domain authority .Getting a back link from a high domain authority site is always valuable.



2. Linking Strategies
-the text in the links should include keywords
- the more inbound links the higher the SE ranking
- if the site linking to us is already indexed, spiders will also receive our site

3. Keywords
- the most important in optimizing rankings
-keywords
balance keyword-rich and readability 4.Title tags

```
    <title>KISSmetrics Web Analytics - Event Tracking, A/B Testing and Conversion
Funnel Software</title>
```

The title tag on pages of your website tells search engines what the page is about. It should be 70 characters or less and include your business or brand name and keywords that relate to that specific page only. This tag is placed between the <HEAD> </HEAD> tags near the top of the HTML code for the page.

5. Meta tag and description tags

-displayed below the title in search results
- use dynamic, promotional language
- use keywords

```
<meta name="description" content="Free Web tutorials on HTML, CSS,
XML, and XHTML">
```

6. Alt tags

-include keywords in your alt tags

<IMG src="star.gif" alt=—star logo">

7. Submit your website to SEs for indexing

-submit your site to search engine directories, directory sites and portal sites
- indexing takes time

SEO techniques are classified into two broad categories:

- **White Hat SEO** - Techniques that search engines recommend as part of a good design.
- **Black Hat SEO** - Techniques that search engines do not approve and attempt to minimize the effect of. These techniques are also known as spamdexing.

White Hat SEO

An SEO tactic is considered as White Hat if it has the following features:

- It conforms to the search engine's guidelines.
- It does not involve in any deception.
- It ensures that the content a search engine indexes, and subsequently ranks, is the same content a user will see.

- It ensures that a web page content should have been created for the users and not just for the search engines.
- It ensures good quality of the web pages.
- It ensures availability of useful content on the web pages.

Always follow a White Hat SEO tactic and do not try to fool your site visitors. Be honest and you will definitely get something more.

Black Hat or Spamdexing

An SEO tactic, is considered as Black Hat or Spamdexing if it has the following features:

- Attempting ranking improvements that are disapproved by the search engines and/or involve deception.
- Redirecting users from a page that is built for search engines to one that is more human friendly.
- Redirecting users to a page that was different from the page the search engine ranked.
- Serving one version of a page to search engine spiders/bots and another version to human visitors. This is called **Cloaking** SEO tactic.
- Using hidden or invisible text or with the page background color, using a tiny font size or hiding them within the HTML code such as "no frame" sections.
- Repeating keywords in the meta-tags, and using keywords that are unrelated to the website content. This is called **meta-tag stuffing**.
- Calculated placement of keywords within a page to raise the keyword count, variety, and density of the page. This is called **keyword stuffing**.
- Creating low-quality web pages that contain very little content but are instead stuffed with very similar keywords and phrases. These pages are called **Doorway or Gateway Pages**.
- Mirror websites by hosting multiple websites - all with conceptually similar content but using different URLs.
- Creating a rogue copy of a popular website which shows contents similar to the original to a web crawler, but redirects web surfers to unrelated or malicious websites. This is called **page hijacking**.

Always stay away from any of the above Black Hat tactics to improve the rank of our site. Search engines are smart enough to identify all the above properties of our site and ultimately we are not going to get anything.

3.5.4 SEO Tools

- Keyword Discovery - Find popular keywords that your site should be targeting.

- Keyword Volume - Estimate how much search traffic a specific keyword receives each month.
- Keyword Density - Analyze how well a specific webpage is targeting a keyword.
- Back link Trackers - Keep track of how many sites are linking to you.
- Site Popularity - Determine how popular your site is on the internet.
- Keyword Rankings - Track of your site's rankings for a keyword in the search engines.
- Firefox Add-ons - Install these add-ons to turn your browser into an SEO research powerhouse.

What are the two major types of SEO

There are two major types of search engine optimization, white hat search engine optimization (the 'good' kind), and black hat (the 'not so good' kind).

SEO Services

There are a number of SEO services which can help contribute to the improvement of the organic search engine rankings of a website. These services include, but are not limited to, on-page (or on-site) optimization, link building, search engine friendly website design and development, and search engine friendly content writing services.

3.6 Web size measurement Issues

of Web size

- The web is really infinite
- Static web contains syntactic duplication, mostly due to mirroring
- Some servers are seldom connected.

3.6.1 Characteristics of Web

Measuring the World Wide Web is a very difficult task due to its dynamic nature. In 1999, there were over 40 million computers in more than 200 countries connected to the Internet. Within these computers, over 3 million of them are Web servers (NetSizer, 1998). There are two explanations why the number of Web servers is huge. The first one is that many Web sites share the same Web server using virtual hosts and not all of them are fully accessible to the outside world. The second one is that not all Web sites start with the prefix www and by only counting Web sites with this prefix there were only 780,000 in 1998.

As for the total number of Web pages, there were estimated to be 350 million in 1998. Between 1997 and 1998, the size of Web pages was doubled in nine months and was gro wing at a rate of 20 million pages per month.(Baeza-Yates, 1999). The most popular formats of Web documents are HTML, followed by GIF and JPG (images format), ASCII files, Postscript and ASP. The most popular compression tools are GNU zip, Zip, and Compress. Most HTML pages are not standard, because they do not comply with HTML specifications. HTML documents seldom start with a document type definition. Also they are typically small with an average of 5 Kb and a median of 2 Kb. On average, each HTML page contains one or two images and five to fifteen hyperlinks. Most of these hyperlinks are local, meaning the associated Web pages are mostly stored in the same Web server (Baeza- Yates, 1999). The top ten most referenced Web sites such as Microsoft, Netscape, and Yahoo are referenced in over 100,000 places. In addition, the site containing the most external links is Yahoo!. Yahoo! glues all the isolated Web sites together to form a large Web database. If we assume that the average size of an HTML page is 5 Kb and there are 300 million Web pages, then we have at least 1.5 terabytes of text. This huge size is consistent with other studies done by other organizations (Baeza-Yates, 1999).

3.7 Search Engine Spam

Search engines make billions of dollars each year selling ads. Most search engine traffic goes to the free, organically listed sites. The ratio of traffic distribution is going to be keyword dependent and search engine dependent, but we believe about 85% of Google‘s traffic clicks on the organic listings. Most other search engines display ads a bit more aggressively than Google does. In many of those search engines, organic listings get around 70% of the traffic. Some sites rank well on merit, while others are there due exclusively to ranking manipulation. In many situations, a proper SEO campaign can provide a much greater ROI(Return on Investments) than paid ads do. This means that while search engine optimizers—known in the industry as SEOs—and search engines have business models that may overlap, they may also compete with one another for ad dollars. Sometimes SEOs and search engines are friends with each other, and, unfortunately, sometimes they are enemies. When search engines return relevant results, they get to deliver more ads. When their results are not relevant, they lose market share. Beyond relevancy, some search engines also try to bias the search results to informational sites such that commercial sites are forced into buying ads. There is a huge sum of money in manipulating search results. There are ways to improve search engine placement that go with the goals of the search engines, and there are also ways that go against them. Quality SEOs aim to be relevant, whether or not they follow search guidelines. Many effective SEO techniques may be considered somewhat spammy. Like anything in life, we should make an informed decision

about which SEO techniques we want to use and which ones we do not .We may choose to use highly aggressive, ―crash and burn‖ techniques, or slower, more predictable, less risky techniques. Most industries will not require extremely aggressive promotional techniques.

3.7.1 How Search Engines Works

Unlike humans, search engines are text-driven. Although technology advances rapidly, search engines are far from intelligent creatures that can feel the beauty of a cool design or enjoy the sounds and movement in movies. Instead, search engines crawl the Web, looking at particular site items (mainly text) to get an idea what a site is about. Search engines perform several activities in order to deliver search results – *crawling, indexing, processing, calculating relevancy*, and *retrieving*.

First, search engines **crawl** the Web to see what is there. This task is performed by a piece of software, called a *crawler* or a *spider* (or Googlebot, as is the case with Google). Spiders follow links from one page to another and index everything they find on their way. Having in mind the number of pages on the Web (over 20 billion), it is impossible for a spider to visit a site daily just to see if a new page has appeared or if an existing page has been modified, sometimes crawlers may not end up visiting our site for a month or two.

After a page is crawled, the next step is to **index** its content. The indexed page is stored in a giant database, from where it can later be retrieved. Essentially, the process of indexing is identifying the words and expressions that best describe the page and assigning the page to particular keywords. For a human it will not be possible to process such amounts of information but generally search engines deal just fine with this task. Sometimes they might not get the meaning of a page right but if we help them by optimizing it, it will be easier for them to classify our pages correctly and for us – to get higher rankings.

When a search request comes, the search engine **processes** it – i.e. it compares the search string in the search request with the indexed pages in the database. Since it is likely that more than one page (practically it is millions of pages) contains the search string, the search engine starts **calculating the relevancy** of each of the pages in its index with the search string.

There are various algorithms to calculate relevancy. Each of these algorithms has different relative weights for common factors like keyword density, links, or meta-tags. That is why different search engines give different search results pages for the same search string. All major search engines, like Yahoo!, Google, Bing, etc. periodically change their algorithms and to keep at the top, we also need to adapt our pages to the latest changes. This is one reason (the other is your competitors) to devote permanent efforts to SEO.

The last step in search engines' activity is **retrieving** the results. Basically, it is nothing more than simply displaying them in the browser – i.e. the endless pages of search results that are sorted from the most relevant to the least relevant sites.

3.7.2 Key Elements to ethical SEO

1. Keyword research

It allows us to see which keywords users actually employ to find products and services within our chosen market, instead of making guesses at the keywords we believe are the most popular.

2. Content development

Content development involves:

- Navigational flow and menu structure
- Site copy or articles
- Headings and sections 3.Web

development

It involves

- Text-based site development wherever possible.
- Clean and logical site structure.
- Proper markup of key page elements.

4. Link Building

Building links will make up about 60% of our work. There are ways to automate this process using shortcuts, workarounds, and submission services.

Internal linking is also very important. Treat the way we link to our own content same as we would link from an external site.

5. Webmaster Tools

Webmaster dashboard is provided by both Google and Bing that gives insight into activity by the search engine on any site that has been registered and verified via dashboard.

Dashboards offer a number of tools which allow us to understand how the search engine sees our site. These are the only way to identify crawling, indexing, and the ranking issue with our site.

3.7.3 How people interact with search engines?

One of the most important elements to building an online marketing strategy around Search Engine Optimization is empathy for our audience.

There are three types of search queries people generally make:

- "Do" Transactional Queries: ―I want to do something, such as buy a plane ticket or listen to a song‖.
- "Know" Informational Queries: ―I need information, such as the name of a band or the best restaurant in New York City‖.
- "Go" Navigation Queries: ―I want to go to a particular place on the Internet, such as Face book or the homepage of the NFL‖.

The search engines' primary responsibility is to serve relevant results to their users. So find what target customers are looking for and make sure the site delivers it to them.

3.8 Web Search Architecture

The main components of a search engine are the crawler, indexer, search index, query engine, and search interface.
A *web crawler* is a software program that traverses web pages, downloads them for indexing, and follows the hyperlinks that are referenced on the downloaded pages a web crawler is also known as a *spider*, a *wanderer* or a *software robot*. The second component is the *indexer* which is responsible for creating the search index from the web pages it receives from the crawler.

The Search Index

The *search index* is a data repository containing all the information the search engine needs to match and retrieve web pages. The type of data structure used to organize the index is known as an *inverted file*. It is very much like an index at the back of a book. It contains all the words appearing in the web pages crawled, listed in alphabetical order (this is called the *index file*), and for each word it has a list of references to the web pages in which the word appears (this is called the *posting list*).

Example:

Consider the entry for ―chess‖ in the search index. Attached to the entry is the posting list of all web pages that contain the word ―chess‖; for example, the entry for ―chess‖ could be chess → [www.chess.co.uk, www.uschess.org,

www.chessclub.com, . . .] Often, more information is stored for each entry in the index such as the number of documents in the posting list for the entry, that is, the number of web pages that contain the keyword, and for each individual entry in the posting file we may also store the number of occurrences of the keyword in the web page and the position of each occurrence within the page. This type of information is useful for determining content relevance.

The search index will also store information pertaining to hyperlinks in a separate *link database*, which allows the search engine to perform hyperlink analysis, which is used as part of the ranking process of web pages. The link database can also be organized as an inverted file in such a way that its index file is populated by URLs and the posting list for each URL entry, called the *source* URL, contains all the *destination* URLs forming links between these source and destination URLs.

The link database for the Web can be used to reconstruct the structure of the web and to have good coverage, its index file will have to contain billions of entries. When we include the posting lists in the calculation of the size of the link database, then the total number of entries in the database will be an order of magnitude higher. Compression of the link database is thus an important issue for search engines, who need to perform efficient hyperlink analysis.

Randall have developed compression techniques for the link database, which take advantage of the structure of the Web. Their techniques are based on the observations that most web pages tend to link to other pages on the same website, and many web pages on the same web site tend to link to a common set of pages. Combing these observations with well-known compression methods, they have managed to reduce the space requirements to six bits per hyperlink.

The text which is attached to a hyperlink, called *link (or anchor) text*, that is clicked on by users following the link, is considered to be part of the web page it references. So when a word such as ―chess‖ appears in some link text, then the posting list for that word will contain an entry for the destination URL of the link.

The Query Engine

of a commercial query engine is a well-guarded secret, since search engines are rightly paranoid, fearing web sites who wish to increase their ranking by unscrupulously taking advantage of the algorithms the search engine uses to rank result pages. Search engines view such manipulation as spam, since it has direct effects on the quality of the results presented to the user.

Spam

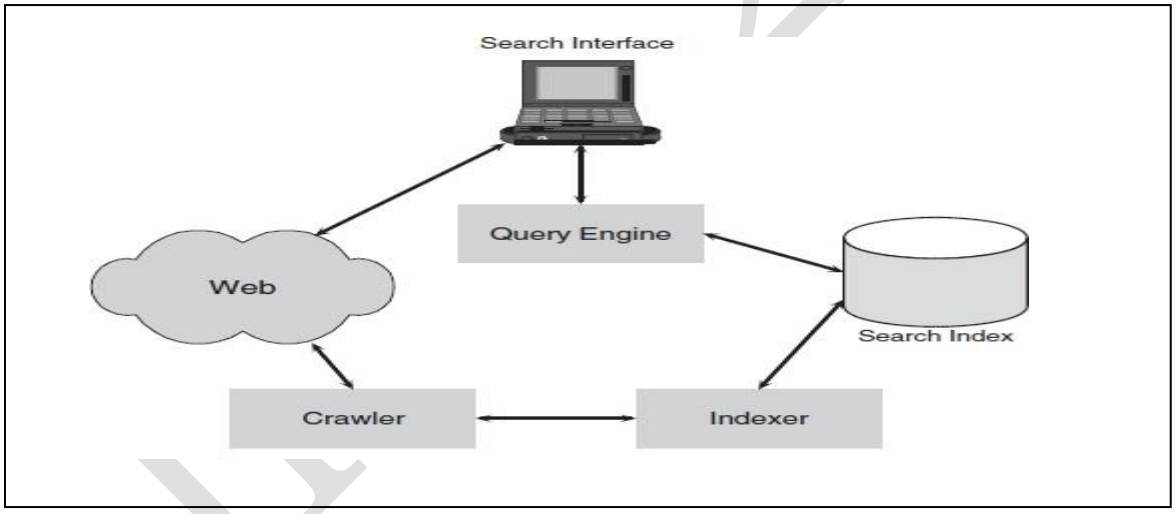
Spam is normally associated with unsolicited e-mail also known as *junk e-mail*, although the word spam originally derives from spiced ham and refers to a (canned meat product.) It is not straightforward to distinguish between search engine spam and organic search engine optimization, where a good and healthy design of web pages leads them to be visible on the top results of search engines for queries related to the pages.

The query engine provides the interface between the search index, the user, and the Web. The query engine processes a user query in two steps. In the first step, the query engine retrieves from the search index information about potentially relevant web pages that match the keywords in the user query, and in the second step a ranking of the results is produced, from the most relevant downwards.
The ranking algorithm combines content relevance of web pages and other relevance measures of web pages based on link analysis and popularity. Deciding how to rank web pages revolves upon our understanding of the concept of what is ―relevant‖ for a user, given a query. The problem with relevance is that what is relevant for one user may not be relevant to another. In a nutshell, relevance is, to a large degree, personal and depends on the context and task the user has in mind. Search engines take a very pragmatic view of relevance and continuously tweak and improve their ranking algorithms by examining how surfers search the Web; for example, by studying recent query logs.

The Search Interface

Once the query is processed, the query engine sends the results list to the *search interface*, which displays the results on the user‘s screen. The user interface provides the look and feel of the search engine, allowing the user to submit queries, browse the results list, and click on chosen web pages for further browsing.

Simplified Search Engine Architecture



3.9 Web Crawling

Crawling is the process of gathering pages from the internet, in order to index them .The objective of crawling is to quickly and efficiently gather as many useful web pages as possible, together with the link structure that interconnects them.

Features of a crawler

- Robustness: ability to handle spider-traps (cycles, dynamic web pages)
- Politeness: policies about the frequency of robot visits
- Distribution: crawling should be distributed within several machines
- Scalability: crawling should be extensible by adding machines, extending bandwidth, etc.
- Efficiency: clever use of the processor, memory, bandwidth
- should detect the most useful pages, to be indexed first
- Freshness: should continuously crawl the web (visiting frequency of a page should be close to its modification frequency)
- Extensibility: should support new data formats (e.g. XML-based formats), new protocols (e.g. ftp), etc.

Crawling process

(a) The crawler begins with a seed set of URLs to fetch

(b) The crawler fetches and parses the corresponding web pages, and extracts both text and links

(c) The text is fed to a text indexer, the links (URL) are added to a URL frontier (crawling agenda)

(d) (continuous crawling) Already fetched URLs are appended to the URL frontier for later re-processing

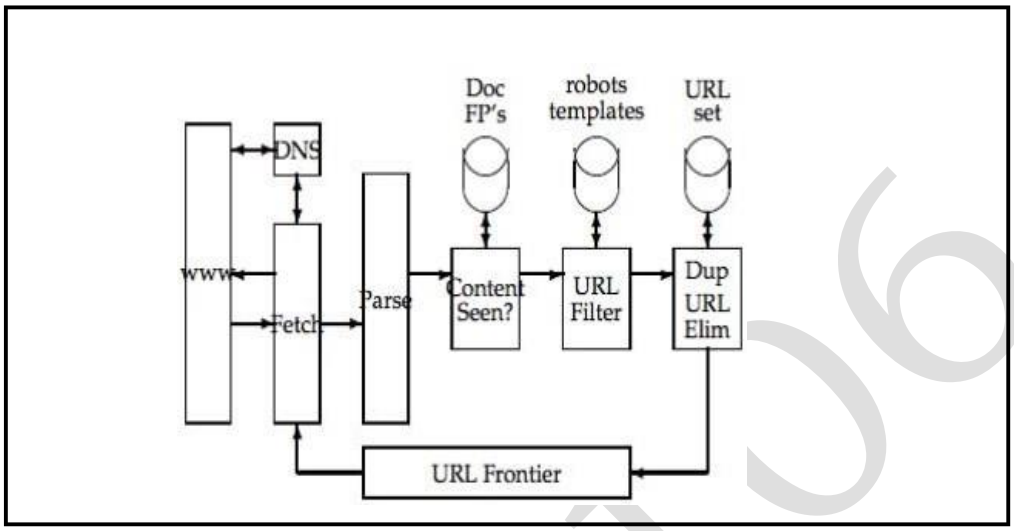
Architecture of a crawler

URL frontier – managing the URLs to be fetched

DNS resolution – determining the host (web server) from which to fetch a page defined by a URL

Fetching module – downloading a remote webpage for processing Parsing module – extracting text and links

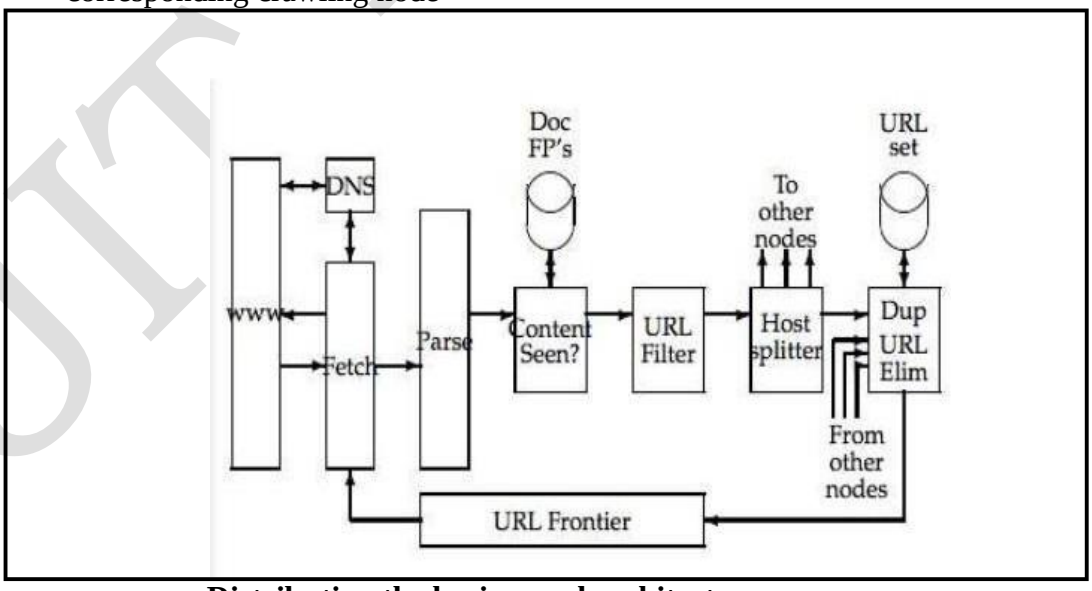
Duplicate elimination – detecting URLs and contents that have been processed a short time



Basic Crawler Architecture

Distributed crawling

- crawling operation can be performed by several dedicated threads
- Parallel crawling can be distributed over nodes of a distributed system (geographical distribution, link-based distribution, etc.)
- Distribution involves a host-splitter, which dispatches URLs to the corresponding crawling node



Distributing the basic crawl architecture

The URL frontier

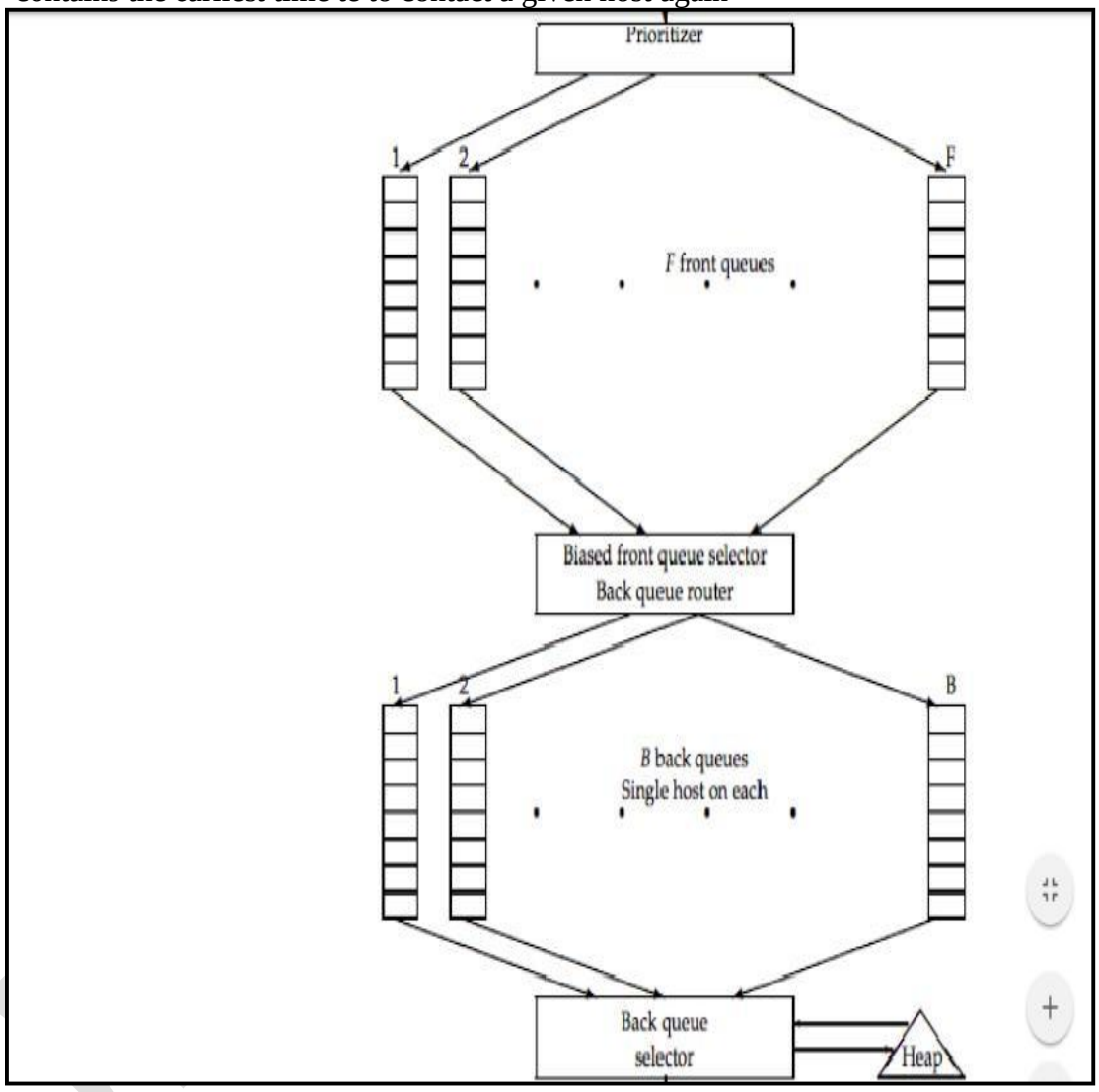
2 types of queues:

- front queues for prioritization

- back queues for politeness

Each back queue only contains URLs for a given host (mapping between hosts and back queue identifiers)

When a back queue is empty, it is filled with URLs from priority queues A heap contains the earliest time te to contact a given host again



URL Frontier

3.9.2 Meta-crawlers

MetaCrawler, a specific search engine owned by InfoSpace, was one of the first metasearch engines. Instead of crawling the Web, metasearch engines search multiple search engines, Internet directory sites, government sites, news sites and aggregate the results. For general search purposes, initiatives such as the Google Knowledge Graph accomplish many of

the same goals as a metasearch engine. For example, a decision about which hotel room to book for

a trip is usually not based on price alone. In fact, the average consumer visits five or six websites before booking a room. Meta-search engines consolidate results from multiple sites and categorize results so consumers can view results based on characteristics other than price.

Eg.http://www.kartoo.com this meta search site shows the results with sites being interconnected by keywords

In specific industries such as the travel industry, metasearch engines are gaining popularity with consumers because they aggregate results across many websites. It provided users with the option to search news, videos, images, white pages, and yellow pages. MetaCrawler blends data from many of the top search engines including Yahoo!, Google, Bing, Ask.com, About.com and others.

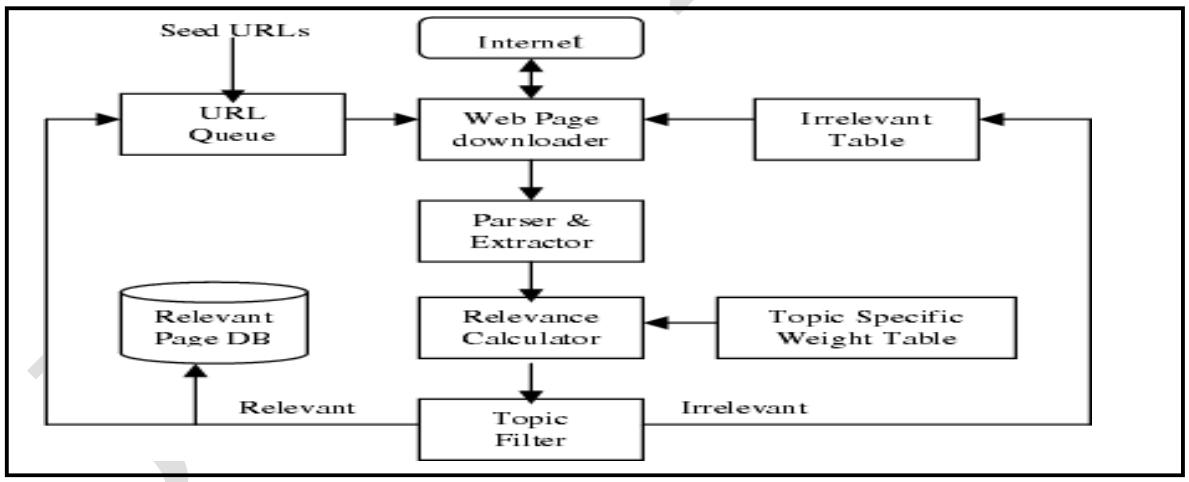
3.10 Focused crawling

Focused crawler is used to collect those web pages that are relevant to a particular topic while filtering out the irrelevant. Thus focused crawling can be used to generate data for an individual user.

There are three major challenges for focused crawling:

(i)It needs to determine the relevance of a retrieved web page. (ii)Predict and identify potential URLs that can lead to relevant pages.

(iii)Rank and order the relevant URLs so the crawler knows exactly what to follow next.



Architecture of Focused crawling

URL queue contains a list of unvisited URLs maintained by the crawler and is initialized with seed URLs. Web page downloader fetches URLs from URL queue and downloads corresponding pages from the internet. The parser and

extractor extracts information such as the terms and the hyperlink URLs from a downloaded page. Relevance calculator calculates relevance of a page w.r.t. topic, and assigns score to URLs extracted from the page. Topic filter analyzes whether the content of parsed pages is related to topic or not. If the page is relevant, the URLs extracted from it will be added to the URL queue, otherwise added to the Irrelevant table.

A focused crawling algorithm loads a page and extracts the links. By rating the links based on keywords the crawler decides which page to retrieve next. The Web is traversed link by link and the existing work is extended in the area of focused document crawling. There are various categories in focused crawlers:

(a) Classic focused crawler
(b) Semantic crawler
(c) Learning crawler

(a) Classic focused crawlers

Guides the search towards interested pages by taking the user query which describes the topic as input. They assign priorities to the links based on the topic of query and the pages with high priority are downloaded first.

These priorities are computed on the basis of similarity between the topic and the page containing the links. Text similarity is computed using an information similarity model such as the Boolean or the Vector Space Model

(b) Semantic crawlers

It is a variation of classic focused crawlers. To compute topic to page relevance downloaded priorities are assigned to pages by applying
semantic similarity criteria, the sharing of conceptually similar terms defines the relevance of a page and the topic. Ontology is used to define the conceptual similarity between the terms.

(c) Learning crawlers

Uses a training process to guide the crawling process and to assign visit priorities to web pages. A learning crawler supplies a training set which consist of relevant and not relevant Web pages in order to train the learning crawler . Links are extracted from web pages by assigning the higher visit priorities to classify relevant topic. Methods based on context graphs
and Hidden Markov Models take into account not only the page content but also the link structure of the Web and the probability that a given page will lead to a relevant page .

3.11 Web indexes

Web indexing means creating indexes for individual Web sites, intranets, collections of HTML documents, or even collections of Web sites. Indexes are systematically arranged items, such as topics or names, that serve as entry points to go directly to desired information within a larger document or set of documents. Indexes are traditionally alphabetically arranged. But they may also make use of hierarchical arrangements, as provided by thesauri, or they

may be entirely hierarchical, as in the case of taxonomies. An index might not even be displayed, if it is incorporated into a searchable database.

Indexing is an analytic process of determining which concepts are worth indexing, what entry labels to use, and how to arrange the entries. As such, Web indexing is best done by individuals skilled in the craft of indexing, either through formal training or through self-taught reading and study.

A Web index is often a brows able list of entries from which the user makes selections, but it may be non-displayed and searched by the user typing into a search box. A site A-Z index is a kind of Web index that resembles an alphabetical back-of-the-book style index, where the index entries are hyperlinked directly to the appropriate Web page or page section, rather than using page numbers. Web indexes work particularly well in sites that have a flat structure with only one or two levels of hierarchy. Indexes complement search engines on larger web sites and for smaller sites, they provide a cost- effective alternative. Whether to use a back-of-the-book style index or a hierarchy of categories will depend on the size of the size of the site and how rapidly the content is changing. Site indexes are best done by individuals skilled in indexing who also have basic skills in HTML or in using HTML indexing tools.

Software can automatically extract page titles or headings, retain their page URL links, and sort them alphabetically. But only a human indexer can edit (or, more precisely, rewrite) such a list of titles and headings into a useful and meaningful set of index entries, add variant terms and cross-references, and decide where and how to structure subentries.

Electronic indexing includes embedded indexing of Word, Framemaker, PDF and InDesign electronic documents of publications, online help and Content Management System tagging. When the pages are edited or changed, the index is regenerated with new page numbers or anchors or URLs with a hyperlink from the index to the relevant page or paragraph.

3.11.1 Difference between Index and Sitemap

A site map is a finding aid organized in the same way as a table of contents, it follows the structure of the site, section by section, instead of being alphabetical. While providing a helpful overview of a site, therefore, site maps do not always enable users to quickly find a specific topic. Also, a site map lists each Web page only once and by its correct name (for example, ―Work For Us‖), without cross-references or variants to make it more easily found. Finally, site maps tend to include only Web pages, and perhaps not even all pages, rather than the additional sections within pages. This is usually dictated by spatial factors: the entire site map should fit on one screen to be most

usable for browsing. An index can be much larger and more detailed, and is not constrained by space, for example, a common convention of indexes is the use of a separate Web page for each letter of the alphabet.

Issues covered by Web index

The Web Index assesses the Web‘s contribution to social, economic and political progress in countries around the world.

The Index measures and ranks:

- **Universal Access**: These sub-index measures whether countries have invested in affordable access to high quality internet infrastructure, as well as investing in the education and skills citizens need to use the Web well.
- **Freedom and Openness:** This sub-index assesses the extent to which citizens enjoy rights to information, opinion, expression, safety and privacy online.
- **Relevant Content and use:** This sub-index maps both Web use by citizens and the content available to citizens in each country, with an emphasis on the extent to which different stakeholders can access information that is relevant to them, in the language that they are most comfortable using and via platforms and channels that are widely available.
- **Empowerment:** This sub-Index aims to assess the difference that the Web is making to people, and the extent to which use of the Web by stakeholders is fostering positive change in four key areas: society, economy, politics and environment.

Web Indexing Examples

- Parliament of Australia
  A large back-of-the-book style index.
- UNIXhelp for Users
  Both a back-of-the-book style index and a searchable index.
- Daily Herald story index - 1901 to 1964
  An alphabetical list of subject headings for newspaper articles.
- Writer's Block
  A "living" index that is updated quarterly.
- The World Bank Group
  Subject list showing major topics and sub-topics. Each page for a major topic is structured differently.
- US Census Bureau
  A large multi-level subject list. Also provides a list of subjects in alphabetical order.

3.12 Near duplicate detection

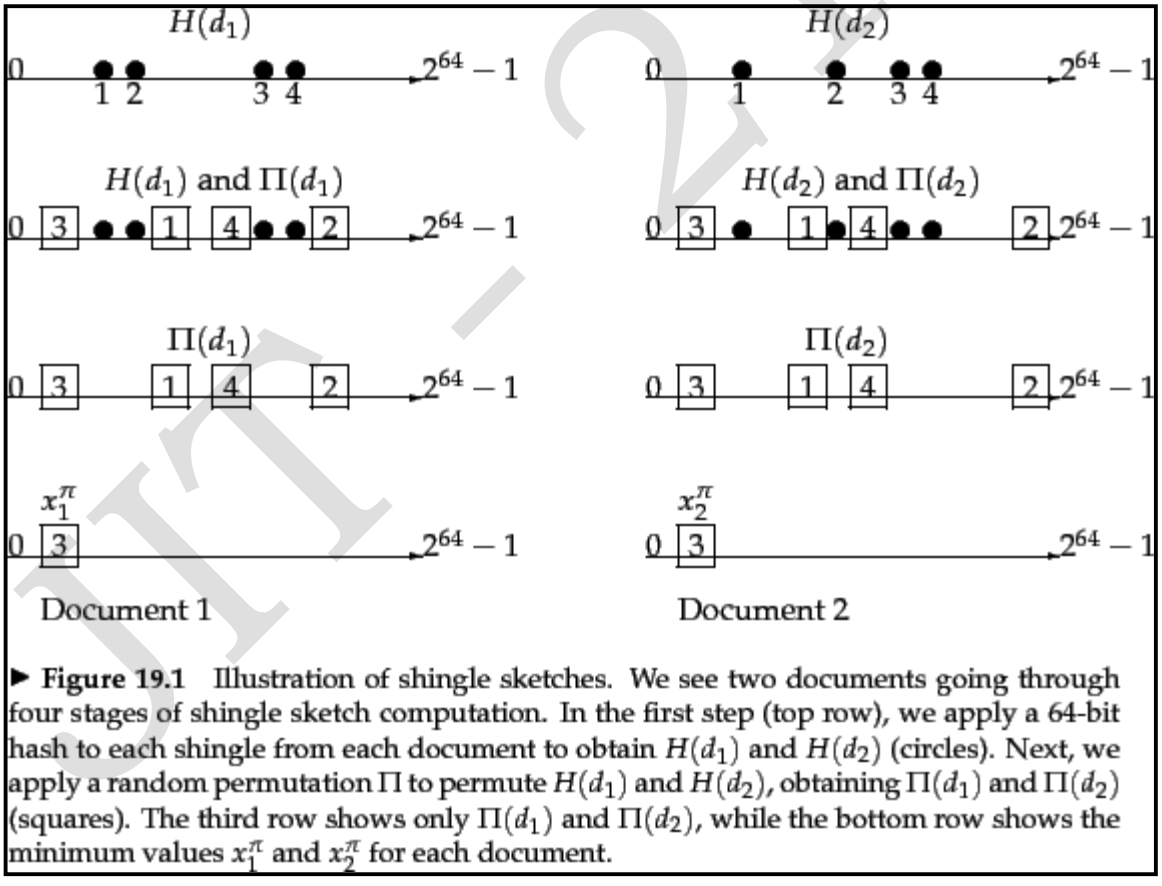
The Web contains multiple copies of the same content. By some estimates, as many as 40% of the pages on the Web are duplicates of other pages. Search engines try to avoid indexing multiple copies of the same content, to keep down storage and processing overheads.

The simplest approach to detecting duplicates is to compute, for each web page, a *fingerprint* that is a succinct (say 64-bit) digest of the characters on that page. Then, whenever the fingerprints of two web pages are equal, we test whether the pages themselves are equal and if so declare one of them to be a duplicate copy of the other. This simplistic approach fails to capture a crucial and widespread phenomenon on the Web: *near duplication*. In many cases, the contents of one web page are identical to those of another except for a few characters - say, a notation showing the date and time at which the page was last modified. Even in such cases, we want to be able to declare the two pages to be close enough that we only index one copy. Short of exhaustively comparing all pairs of web pages, an infeasible task at the scale of billions of pages, how can we detect and filter out such near duplicates?

We now describe a solution to the problem of detecting near-duplicate web pages. The answer lies in a technique known as *shingling* . Given a positive integer k and a sequence of terms in a document d, define the k-shingles of d to be the set of all consecutive sequences of k terms in d. As an example, consider the following text: a rose is a rose is a rose. The 4-shingles for this text ($k = 4$ is a typical value used in the detection of near-duplicate web pages) are a rose is a, rose is a rose and is a rose is. The first two of these shingles each occur twice in the text. Intuitively, two documents are near duplicates if the sets of shingles generated from them are nearly the same. We now make this intuition precise, then develop a method for efficiently computing and comparing the sets of shingles for all web pages. Let $S(d_j)$ denote the set of shingles of document d_j. Recall the *Jaccard coefficient* , which measures the degree of overlap between the sets $S(d_1)$ and $S(d_2)$ as $|S(d_1) \cap S(d_2)|/|S(d_1) \cup S(d_2)|$; denote this by $J(S(d_1), S(d_2))$. Our test for near duplication between d_1 and d_2 is to compute this Jaccard coefficient, if it exceeds a preset threshold (say, 0.9), we declare them near duplicates and

eliminate one from indexing. However, this does not appear to have simplified matters: we still have to compute Jaccard coefficients pairwise.

To avoid this, we use a form of hashing. First, we map every shingle into a hash value over a large space, say 64 bits. For $j = 1,2$, let $H(d_j)$ be the corresponding set of 64-bit hash values derived from $S(d_j)$. We now invoke the following trick to detect document pairs whose sets $H()$ have large Jaccard overlaps. Let π be a random permutation from the 64-bit integers to the 64-bit integers. Denote by $\Pi(d_j)$ the set of permuted hash values in $H(d_j)$; thus for each $h \in H(d_j)$, there is a corresponding value $\pi(h) \in \Pi(d_j)$.



► **Figure 19.1** Illustration of shingle sketches. We see two documents going through four stages of shingle sketch computation. In the first step (top row), we apply a 64-bit hash to each shingle from each document to obtain $H(d_1)$ and $H(d_2)$ (circles). Next, we apply a random permutation Π to permute $H(d_1)$ and $H(d_2)$, obtaining $\Pi(d_1)$ and $\Pi(d_2)$ (squares). The third row shows only $\Pi(d_1)$ and $\Pi(d_2)$, while the bottom row shows the minimum values x_1^π and x_2^π for each document.

Let x_j^π be the smallest integer in $\Pi(d_j)$. Then

Theorem.

$$J(S(d_1), S(d_2)) = P(x_1^\pi = x_2^\pi). \tag{247}$$

Proof. We give the proof in a slightly more general setting: consider a family of sets whose elements are drawn from a common universe. View the sets as columns of a matrix A, with one row for each element in the universe. The element $a_{ij} = 1$ if element i is present in the set S_j that the jth column represents.Let Π be a random permutation of the rows of A; denote by $\Pi(S_j)$ the column that results from applying Π to the jth column. Finally, let x_j^π be the index of the first row in which the column $\Pi(S_j)$ has a 1. We then prove that for any two columns j_1, j_2,

$$P(x_{j_1}^\pi = x_{j_2}^\pi) = J(S_{j_1}, S_{j_2}). \tag{248}$$

If we can prove this, the theorem follows.

S_{j_1}	S_{j_2}
0	1
1	0
1	1
0	0
1	1
0	1

Figure 19.9: Two sets S_{j_1} and S_{j_2}; their Jaccard coefficient is $2/5$.

Consider two columns j_1, j_2 as shown in Figure 19.9 . The ordered pairs of entries of S_{j_1} and S_{j_2} partition the rows into four types: those with 0's in both of these columns, those with a 0 in S_{j_1} and a 1 in S_{j_2}, those with a 1 in S_{j_1} and a 0 in S_{j_2}, and finally those with 1's in both of these columns. Indeed, the first four rows of Figure 19.9 exemplify all of these four types of rows. Denote by C_{00} the number of rows with 0's in both columns, C_{01} the second, C_{10} the third and C_{11} the fourth. Then,

$$J(S_{j_1}, S_{j_2}) = \frac{C_{11}}{C_{01} + C_{10} + C_{11}}. \quad (249)$$

To complete the proof by showing that the right-hand side of Equation 249 equals $P(x_{j_1}^{\pi} = x_{j_2}^{\pi})$, consider scanning columns j_1, j_2 in increasing row index until the first non-zero entry is found in either column. Because Π is a random permutation, the probability that this smallest row has a 1 in both columns is exactly the right-hand side of Equation 249

3.13 Index Compression

Lossy and Losseless compression

- Lossy Compression that involves the removal of data.
- Loseless Compression that involves no removal of data.

3.13.1

The dictionary and the inverted index as the central data structures in information retrieval (IR). There are two more subtle benefits of compression.

The first is increased use of caching. Search systems use some parts of the dictionary and the index much more than others. For example, if we cache the postings list of a frequently used query term *t*, then the computations necessary for responding to the one-term query *t* can be entirely done in memory. With compression, we can fit a lot more information into main memory. Instead of having to expend a disk seek when processing a query with *t*, we instead access its postings list in memory and decompress it. As we will see below, there are simple and efficient decompression methods, so that the penalty of having to decompress the postings list is small. As a result, we are able to decrease the response time of the IR system substantially. Because memory is a more expensive resource than disk space, increased speed owing to caching – rather than decreased space requirements – is often the prime motivator for compression.

The second more subtle advantage of compression is faster transfer of data from disk to memory. Efficient decompression algorithms run so fast on modern hardware that the total time of transferring a compressed chunk of data from disk and then decompressing it is usually less than transferring the same chunk of data in uncompressed form. For instance, we can reduce

input/output (I/O) time by loading a much smaller compressed postings list, even when you add on the cost of decompression. So, in most cases, the retrieval system runs faster on compressed postings lists than on uncompressed postings lists.

3.13.2 Dictionary compression

One of the primary factors in determining the response time of an IR system is the number of disk seeks necessary to process a query. If parts of the dictionary are on disk, then many more disk seeks are necessary in query evaluation. Thus, the main goal of compressing the dictionary is to fit it in main memory, or at least a large portion of it, to support high query throughput. Although dictionaries of very large collections fit into the memory of a standard desktop machine, this is not true of many other application scenarios.

For example, an enterprise search server for a large corporation may have to index a multi terabyte collection with a comparatively large vocabulary because of the presence of documents in many different languages. We also want to be able to design search systems for limited hardware such as mobile phones and onboard computers. Other reasons for wanting to conserve memory are fast startup time and having to share resources with other applications. The search system on your PC must get along with the memory- hogging word processing suite you are using at the same time.

Example considered: the Reuters-RCV1 collection

size of	word types dictionary size	Δ	cumul.	non-positional postings non-positional index size	Δ	cumul.	positional postings (word tokens) positional index size	Δ
unfiltered	484,494			109,971,179			197,879,290	
no numbers	473,723	-2%	-2%	100,680,242	-8%	-8%	179,158,204	-9%
case folding	391,523	-17%	-19%	96,969,056	-3%	-12%	179,158,204	-0%
30 stop words	391,493	-0%	-19%	83,390,443	-14%	-24%	121,857,825	-31%
150 stop words	391,373	-0%	-19%	67,001,847	-30%	-39%	94,516,599	-47%
stemming	322,383	-17%	-33%	63,812,300	-4%	-42%	94,516,599	-0%

	term	document frequency	pointer to postings list
	a	656,265	⟶
	aachen	65	⟶
	…	…	…
	zulu	221	⟶
space needed:	20 bytes	4 bytes	4 bytes

Figure Storing the dictionary as an array of fixed-width entries.

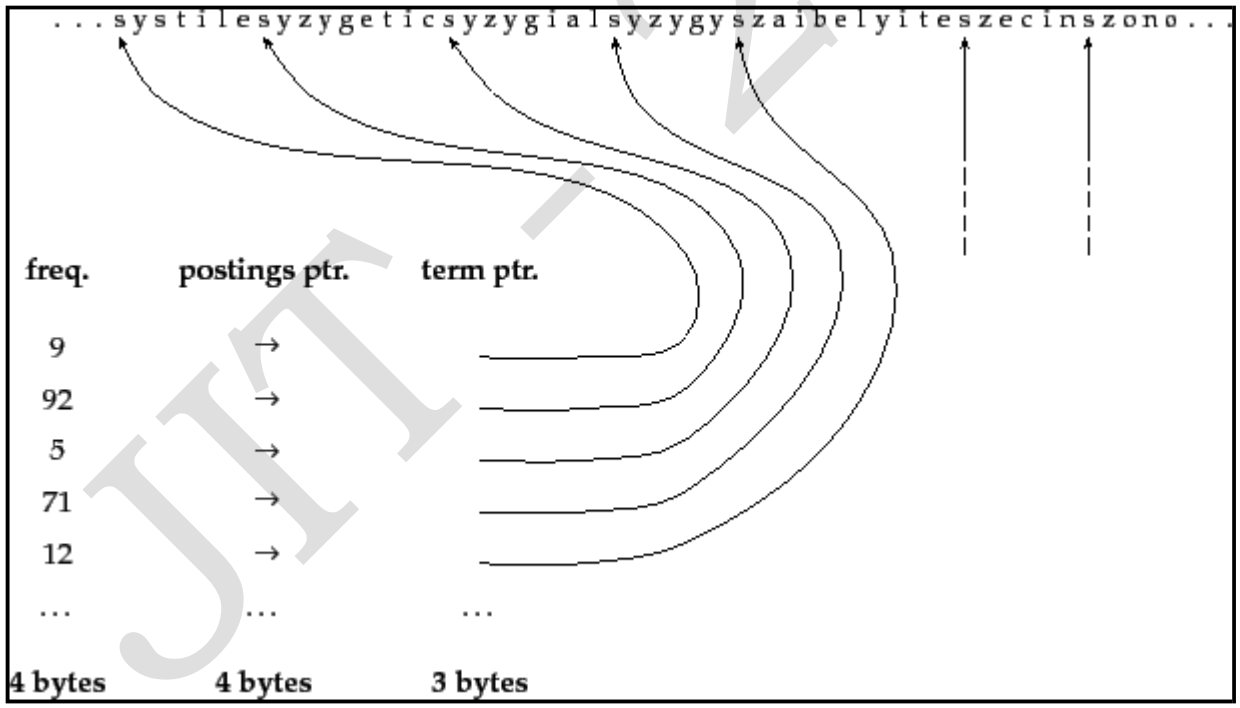
Total space:
M×(2×20 + 4 + 4) = 400,000×48 = 19.2 MB
NB: why 40 bytes per term ? (unicode + max. length of a term)

3.13.3 Dictionary as a string

The simplest data structure for the dictionary is to sort the vocabulary lexicographically and store it in an array of fixed-width entries as shown in Figure above . Assuming a Unicode representation, we allocate 2*20 We allocate 20 bytes for the term itself (because few terms have more than twenty characters in English), 4 bytes for its document frequency, and 4 bytes for the pointer to its postings list. Four-byte pointers resolve a 4 gigabytes (GB) address space. For large collections like the web, we need to allocate more bytes per pointer. We look up terms in the array by binary search.

For Reuters-RCV1, we need M *(2*20+4+4)=400,000 * 48 =19.2 megabytes (MB)

M*(20+4+4)=400,000*28=11.2 megabytes(MB) for storing the dictionary in this scheme.



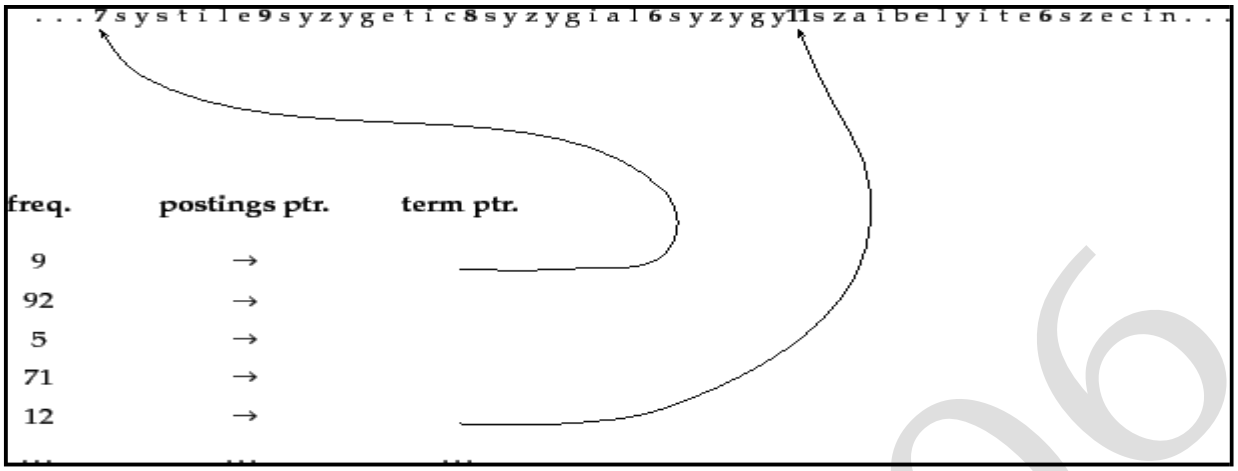
Dictionary-as-a-string storage

- 4 bytes per term for frequency
- 4 bytes per term for pointer to postings list
- 3 bytes per pointer into string (need $\log_2 400000 \approx 22$ bits to resolve 400,000 positions)
- 8 chars (on average) for term in string
- Space: $400{,}000 \times (4 + 4 + 3 + 2 \times 8) = 10.8$ MB

Space use of Dictionary as-a-string

Pointers mark the end of the preceding term and the beginning of the next. For example, the first three terms in this example are systile, syzygetic, and syzygial. Using fixed-width entries for terms is clearly wasteful. The average length of a term in English is about eight characters icompresstb1, so on average we are wasting twelve characters (or 24 bytes) in the fixed-width scheme. Also, there is no way of storing terms with more than twenty characters like hydrochlorofluorocarbons and supercalifragilisticexpialidocious. We can overcome these shortcomings by storing the dictionary terms as one long string of characters, as shown in above Figure . The pointer to the next term is also used to demarcate the end of the current term. As before, locating terms in the data structure by way of binary search in the (now smaller) table. This scheme saves us 60% compared to fixed-width storage - 24 bytes on average of the 40 bytes 12 bytes on average of the 20 bytes we allocated for terms before. However, we now also need to store term pointers. The term pointers resolve 400,000*8=3.2*106 positions, so they need to be log23.2*106 ≈22 bits or 3 bytes long.

In this new scheme, user needs 400,000 * (4+4+3+8)=7.6 MB space for the Reuters-RCV1 dictionary: 4 bytes each for frequency and postings pointer, 3 bytes for the term pointer, and 8 bytes on average for the term. So user have reduced the space requirements by one third from 19.211.2 to 10.87.6 MB.



Blocked storage with four terms per block.

The first block consists of systile, syzygetic, syzygial, and syzygy with lengths of seven, nine, eight, and six characters, respectively. Each term is preceded by a byte encoding its length that indicates how many bytes to skip to reach subsequent terms.

One block in blocked compression ($k = 4$) ...

8 a u t o m a t a **8** a u t o m a t e **9** a u t o m a t i c **10** a u t o m a t i o n

⇓

... further compressed with front coding.

8 a u t o m a t ∗ a **1** ◇ e **2** ◇ i c **3** ◇ i o n

- End of prefix marked by ∗
- Deletion of prefix marked by ◇

Space use for block storage in front coding

- Let us consider blocks of size k
- We remove $k-1$ pointers, but add k bytes for term length
- Example: $k = 4$, $(k-1) \times 3$ bytes saved (pointers), and 4 bytes added (term length) $\rightarrow$ 5 bytes saved
- Space saved: $400{,}000 \times (1/4) \times 5 = 0.5$ MB (dictionary reduced to 10.3 MB)
- Why not taking $k > 4$?

Dictionary compression for Reuters

Representation dictionary,	size in MB
fixed-width	19.2
dictionary as a string	10.8
~, with blocking, k= 4	10.3
~, with blocking & front coding	7.9

3.13.4 Postings file compression/ the inverted index Inverted file creation

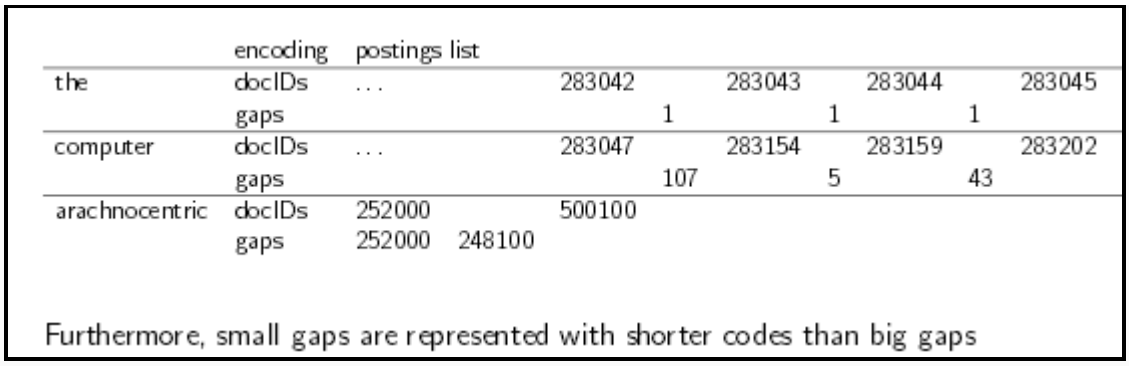
1. First pass over the collection to determine the number of unique terms (vocabulary) and the number of documents to be indexed

2. Allocate the matrix and second pass over the collection to fill the matrix 3.Traverse the matrix row by row and write posting lists to file

Postings compression

GAP ENCODING

Encoding gaps instead of document IDs. For example, we store gaps 107, 5, 43, ..., instead of docIDs 283154, 283159, 283202, ... for computer. The first docID is left unchanged (only shown for arachnocentric)



	encoding	postings list							
the	docIDs	. . .		283042		283043		283044	283045
	gaps				1		1	1	
computer	docIDs	. . .		283047		283154		283159	283202
	gaps				107		5	43	
arachnocentric	docIDs	252000		500100					
	gaps	252000	248100						

Furthermore, small gaps are represented with shorter codes than big gaps

Three posting entries

The postings file is much larger than the dictionary, factor of at least 10.

- Key desideratum: store each posting compactly.
- A posting for our purposes is a docID.
- For Reuters(8,00,000 documents) we would use 32 bit per docID when using 4 byte in integers.
- Alternatively, we can use log 2800,000≈ 20bits per docID.
- Our goal: use far fewer than 20bits per docID.

Postings: Two conflicting forces

- A term like arachnocentric occurs in may be one doc out of a million– user would like to store this posting using log21M~20bits.
- A term like the occurs in virtually every doc, so 20bits/posting is too expensive.
- Prefer 0/1 bitmap vector in this case.

Postings file entry

We store the list of docs containing a term in increasing order of docID.

- computer
- 33,47,154,159,202
- Consequence: it suffices to store gaps
- 33,14,107,5,43

Hope: most gaps can be encoded/stored with far fewer than 20 bi

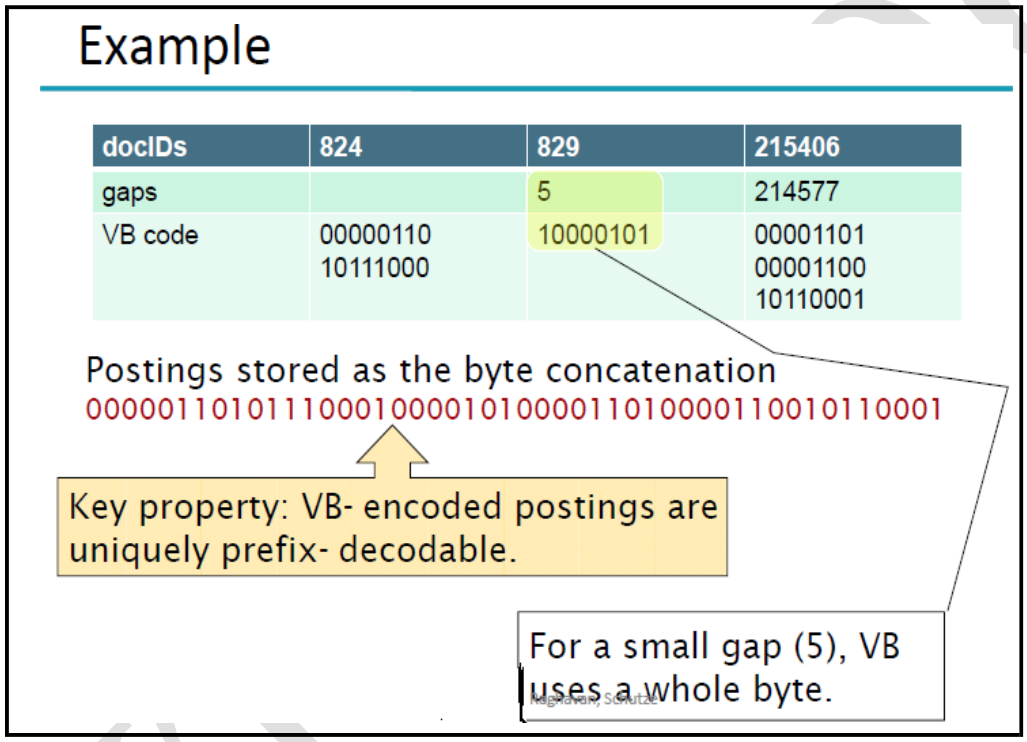
Variable length encoding

- For arachnocentric, use ~20 bits/gap entry.
- For the, use ~1 bit/gap entry.
- If the average gap for a term is G, use ~log2G bits/gap entry.
- Encode every integer (gap) with about as few bits as needed for that integer
- This requires a variable length encoding

- Variable length codes achieve this by using short codes for small numbers

Variable Byte (VB) codes

- For a gap value G, we want to use close to the fewest bytes needed to hold log2 G bits
- Begin with one byte to store G and dedicate 1 bit in it to be a continuation bit c
- If G ≤127, binary encode it in the 7 available bits and set c =1 else encode G's lower order 7 bits and then use additional bytes to encode the higher order bits using the same algorithm
- At the end set the continuation bit of the last byte to 1 (c =1) – and for the other bytes c = 0.



Other variable unit codes

Instead of bytes, we can also use a different ―unit of alignment‖: 32 bits (words), 16 bits, 4 bits (nibbles). Variable byte alignment wastes space if we have many small gaps – nibbles do better in such cases.

Unary code

Represent n as n 1s with a final 0.

- Unary code for 3 is 1110.
- Unary code for 40 is 11111111111111111111111111111111111111111110 .

 Unary code for 80 is: 11111111111111111111111111111111111111111111
11111111111111111111111111111111111110

Gamma codes

We can compress better with bit level codes

- The Gamma code is the best known of these.
- Represent a gap G as a pair length and offset
- offset is G in binary, with the leading bit cut off For example
  13 → 1101 → 101
- length is the length of offset For 13
  (offset 101), this is 3.
- We encode length with unary code: 1110.
- Gamma code of 13 is the concatenation of length and offset: 1110101

Gamma code examples

number	length	offset	γ-code
0			none
1	0		0
2	10	0	10,0
3	10	1	10,1
4	110	00	110,00
9	1110	001	1110,001
13	1110	101	1110,101
24	11110	1000	11110,1000
511	111111110	11111111	111111110,11111111
1025	11111111110	0000000001	11111111110,0000000001

Gamma code properties

G is encoded using 2 ⌊log G⌋ + 1 bits

- Length of offset is ⌊log G⌋ bits
- Length of length is ⌊log G⌋ + 1bits

All gamma codes have an odd number of bits

- Almost within a factor of 2 of best possible, log2 GGamma

code is uniquely prefix-decodable, like VB

- Gamma code can be used for any distribution
- Gamma code is parameter-free

3.14 XML Retrieval

What is XML?

A meta-language (language to describe other languages).XML is able to represent a mix of structured and text(unstructured) information, defined by WWW Consortium headed by James Clark. It is the de facto standard markup language. Example :

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

Fig 1 An XML Document

An XML document is an ordered, labeled tree. Each node of the tree is an *XML element* and is written with an opening and closing *tag* . An element can have one or more *XML attributes* . In the XML document in Figure above , the scene element is enclosed by the two tags <scene ...> and </scene>. It has an attribute number with value vii and two child elements, title and verse.

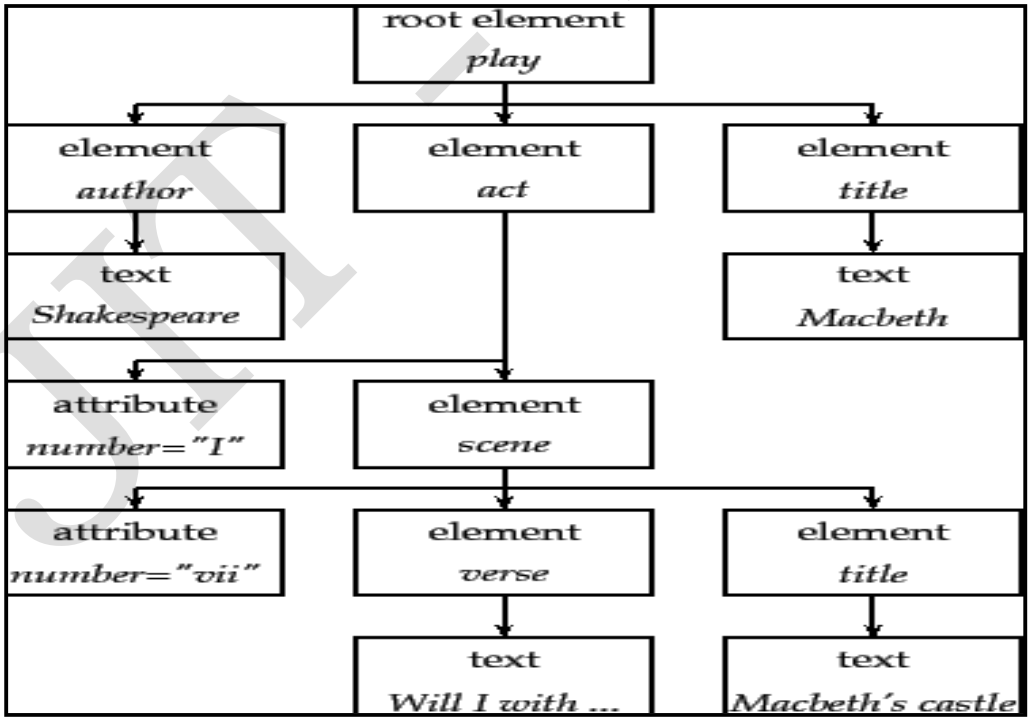


Fig 2:The XML document in Figure 1 as a simplified DOM object.

Figure 1 shows Figure 2 as a tree. The *leaf nodes* of the tree consist of text, e.g., Shakespeare, Macbeth, and Macbeth's castle. The tree's *internal nodes* encode either the structure of the document (title, act, and scene) or *metadata* functions (author).

The standard for accessing and processing XML documents is the XML Document Object Model or *DOM* . The DOM represents elements, attributes and text within elements as nodes in a tree. Figure 2 is a simplified DOM representation of the XML document in Figure 1 . With a DOM API, we can process an XML document by starting at the root element and then descending down the tree from parents to children.

3.14.1 Compare XML vs HTML

XML	HTML
Extensible set of tags	Fixed set of tags
Content oriented	Presentation oriented
Standard Data infrastructure	No data validation capabilities
Allows multiple output forms	

XML is an important standard for both exchanging data between applications and encoding documents. To support this more data-oriented view, the database community has defined languages for describing the structure of XML data (*XML Schema*), and querying and manipulating that data (*XQuery* and *XPath*).

XQuery is a query language that is similar to the database language SQL, with the major difference that it must handle the hierarchical structure of XML data instead of the simpler tabular structure of relational data.

XPath is a subset of XQuery that is used to specify the search constraints for a single type of XML data or document. XPath or *XPath* is a standard for enumerating paths in an XML document collection. Example, could be used in an XML movie database to find the movies that were directed by a particular person and were released in a particular year.

The XPath expression node selects all nodes of that name. Successive elements of a path are separated by slashes, so act/scene selects all scene elements whose parent is an act element. Double slashes indicate that an arbitrary number of elements can intervene on a path: play//scene selects all

scene elements occurring in a play element. In Figure 2 this set consists of a single scene element, which is accessible via the path play, act, scene from the top. An initial slash starts the path at the root element. /play/title selects the play's title in Figure 1 , /play//title selects a set with two members (the play's title and the scene's title), and /scene/title selects no elements. For notational convenience, we allow the final element of a path to be a vocabulary term and separate it from the element path by the symbol #, even though this does not conform to the XPath standard. For example, title#"Macbeth" selects all titles containing the term Macbeth.

XQuery could be used to combine information about movies with information about actors, assuming that the XML movie database contained both movie and actor ―documents.‖ An example would be to find movies starring actors who were born in Australia.

A schema puts constraints on the structure of allowable XML documents for a particular application. A schema for Shakespeare's plays may stipulate that scenes can only occur as children of acts and that only acts and scenes have the number attribute. Two standards for schemas for XML documents are *XML DTD* (document type definition) and *XML Schema*. Users can only write structured queries for an XML retrieval system if they have some minimal knowledge about the schema of the collection.

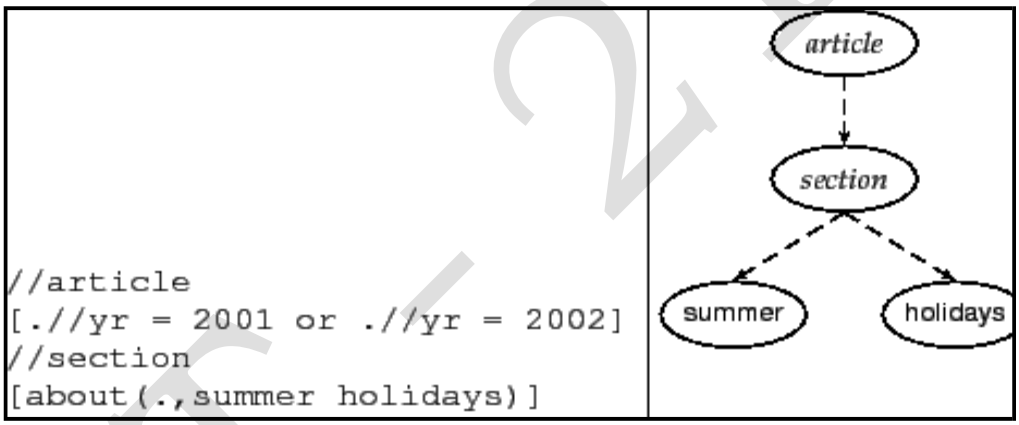


Fig 3: An XML query in NEXI format and its partial representation as a tree.

A common format for XML queries is *NEXI* (Narrowed Extended XPath I). We give an example in Figure 3. We display the query on four lines for typographical convenience, but it is intended to be read as one unit without line breaks. In particular, //section is embedded under //article.

The query in Figure 3 specifies a search for sections about the summer holidays that are part of articles from 2001 or 2002. As in XPath double

slashes indicate that an arbitrary number of elements can intervene on a path. The dot in a clause in square brackets refers to the element the clause modifies. The clause [.//yr = 2001 or .//yr = 2002] modifies //article. Thus, the dot refers to //article in this case. Similarly, the dot in [about(., summer holidays)] refers to the section that the clause modifies.

The two yr conditions are relational attribute constraints. Only articles whose yr attribute is 2001 or 2002 (or that contain an element whose yr attribute is 2001 or 2002) are to be considered. The about clause is a ranking constraint: Sections that occur in the right type of article are to be ranked according to how relevant they are to the topic summer holidays.

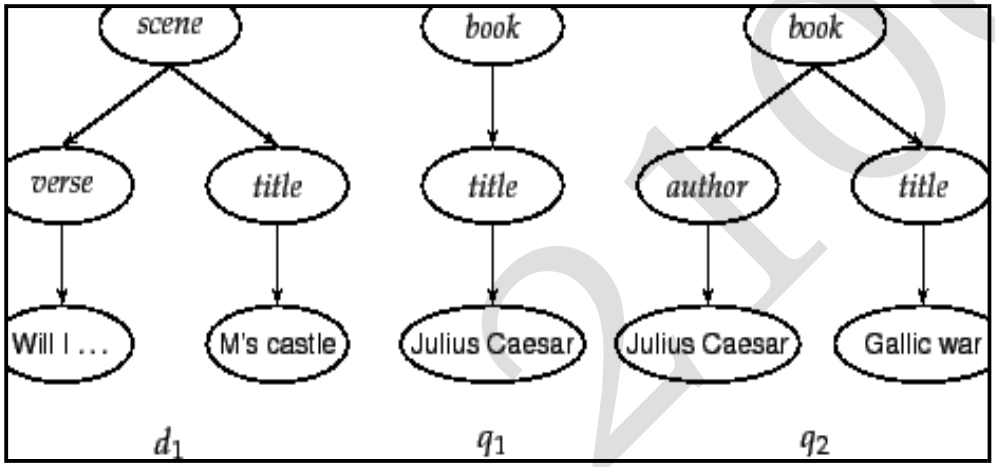


Fig 4Tree representation of XML documents and queries.

User focus on the core information retrieval problem in XML retrieval, namely how to rank documents according to the relevance criteria expressed in the about conditions of the NEXI query.

If user discard relational attributes, it can represent documents as trees with only one type of node: element nodes. In other words, remove all attribute nodes from the XML document, such as the number attribute in Figure 1 . Figure 4 shows a subtree of the document in Figure 1 as an element-node tree (labeled d1).We can represent queries as trees in the same way. This is a *query- by-example* approach to query language design because users pose queries by creating objects that satisfy the same formal description as documents. In Figure 4 , q^1 is a search for books whose titles score highly for the keywords Julius Caesar. q^2 is a search for books whose author elements score highly for Julius Caesar and whose title elements score highly for Gallic war.

3.14.2 Challenges in XML retrieval

The first challenge in structured retrieval is that users want us to return parts of documents (i.e., XML elements), not entire documents as IR systems usually do in unstructured retrieval. If we query Shakespeare's plays for Macbeth's castle, should we return the scene, the act or the entire play in Figure 2 ? In this case, the user is probably looking for the scene. On the other hand, an otherwise unspecified search for Macbeth should return the play of this name, not a sub unit. One criterion for selecting the most appropriate part of a document is the structured document retrieval principle

Structured document retrieval principle. A system should always retrieve the most specific part of a document answering the query. Motivates a retrieval strategy that returns the smallest unit that contains the information sought, but does not go below this level.

Eg.Consider the query title#"Macbeth" applied to Figure 2 . The title of the tragedy, Macbeth, and the title of Act I, Scene vii, Macbeth's castle, are both good hits because they contain the matching term Macbeth. But in this case, the title of the tragedy, the higher node, is preferred. Deciding which level of the tree is right for answering a query is difficult.

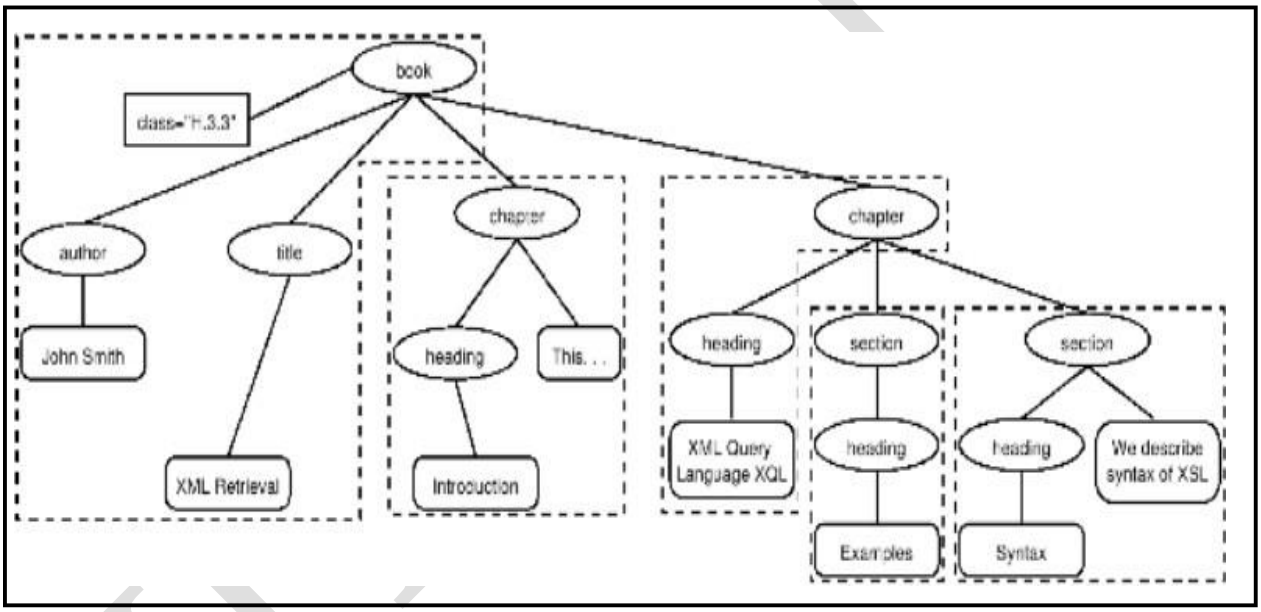


Fig 1.Partitioning an XML document into non-overlapping indexing units

The second challenge is the issue of which parts of a document to index. The Central notion for indexing and ranking in IR is documents unit or indexing unit. In unstructured retrieval, it is usually straightforward: files on the desktop, email ,messages, web pages on the web etc. In structured retrieval, there are four main different approaches in defining the indexing unit

- non-overlapping pseudo documents
- top down
- bottom up
- all

In approach 1 Grouping nodes into non-overlapping pseudo documents as in Fig 1.In the above example Fig Indexing units are books, chapters, section, but without overlap. The main disadvantage is pseudo-documents may not make sense to the user because they are not coherent units. In approach 2 Top down (2-stage process) 1)Start with one of the latest elements as the indexing unit, eg, the book element in a collection of books 2)Then, post-process search results to find for each book the sub-element that is the best hit. This two- stage retrieval process often fails to return the best sub-element because the relevance of a whole book is often not a good predictor of the relevance of small sub-elements within it. In approach 3 Bottom up ,instead of retrieving large units and identifying sub-elements (top down), we can search all leaves, select the most relevant ones and then extend them to larger units in post-processing Similarly in top down, the relevance of a leaf element is often not a good predictor of the relevance of elements it is contained in. In approach 4 index all elements, which is the least restrictive approach. Many XML elements are not meaningful search results, e.g., an ISBN number. Indexing all elements means that search results will be highly redundant. Eg. For the query Macbeth‘s castle we would return all of the play, act, scene and stage elements on the path between the root node and Macbeth‘s castle. The leaf node would then occur 4 times in the result set: 1 directly and 3 as part of other elements.

We call elements that are contained within each other nested elements.

Returning redundant nested elements in a list of returned hits is not very user-friendly. The third challenge is nested elements. Because of the redundancy caused by the nested elements it is common to restrict the set of elements eligible for retrieval. Restriction strategies include:

- discard all small elements
- discard all element types that users do not look at (working XML retrieval system logs)
- discard all element types that assessors generally do not judge to be relevant (if relevance assessments are available)
- only keep element types that a system designer or librarian has deemed to be useful search results .In most of these approaches, result sets will still contain all nested elements.

Further techniques are to

- remove nested elements in a post-processing step to reduce redundancy
- collapse several nested elements in the results list and use highlighting of query terms to draw user‘s attention to the relevant passages.

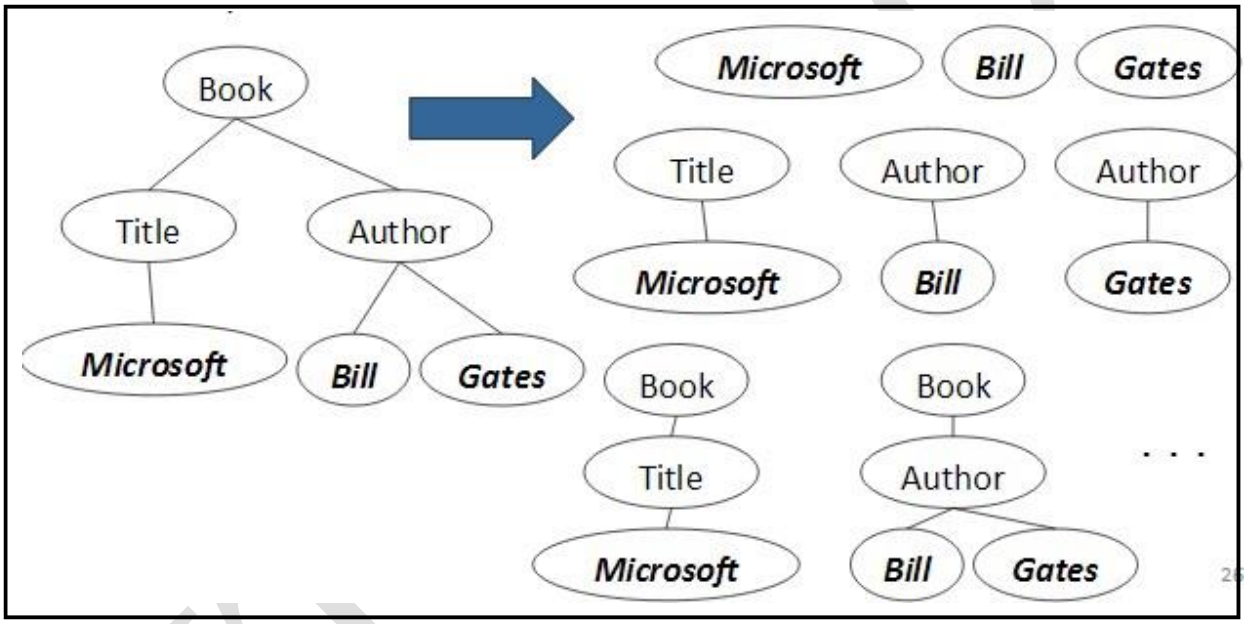
Highlighting:

Gain 1: enables users to scan medium-sized elements (e.g., a section) thus, if the section and the paragraph both occur in the results list, it is sufficient to show the section.

Gain 2: paragraphs are presented in-context (i.e., their embedding section). This context may be helpful in interpreting the paragraph.

Vector space model for XML retrieval

To have each dimension of the vector space encode a word together with its position within the XML tree.



A mapping of an XML document(left) to a set of lexicalized sub trees(right)

- Take each text node (leaf) and break it into multiple nodes, one for each word. E.g. split Bill Gates into Bill and Gates
- Define the dimensions of the vector space to be lexicalized sub trees of documents – sub trees that contain at least one vocabulary term.

Here we represent queries and documents as vectors in this space of lexicalized sub trees and compute matches between them,

e.g. using the vector space formalism. The main difference is that the dimensions of vector space in unstructured retrieval are vocabulary terms whereas they are lexicalized Sub trees in XML retrieval.

Structural term

There is a tradeoff between the dimensionality of the space and the accuracy of query results.

- To restrict dimensions to vocabulary terms, then we have a standard vector space retrieval system that will retrieve many documents that do not match the structure of the query (e.g., Gates in the Title as opposed to the author element).
- To create a separate dimension for each lexicalized sub tree occurring in the collection, the dimensionality of the space becomes too large.

To compromise this index all paths that end in a single vocabulary term, in other words all XML-context term pairs. We call such an XML-context term pair a structural term and denote it by <c,t>: a pair of XML-context c and vocabulary term t.

Context resemblance

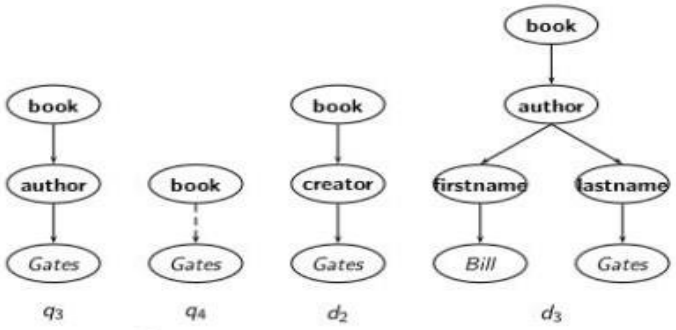
A simple measure of the similarity of a path cq in a query and a path cd in a document is the following context resemblance function CR:

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

$|c_q|$ and $|c_d|$ are the number of nodes in the query path and document path, resp.

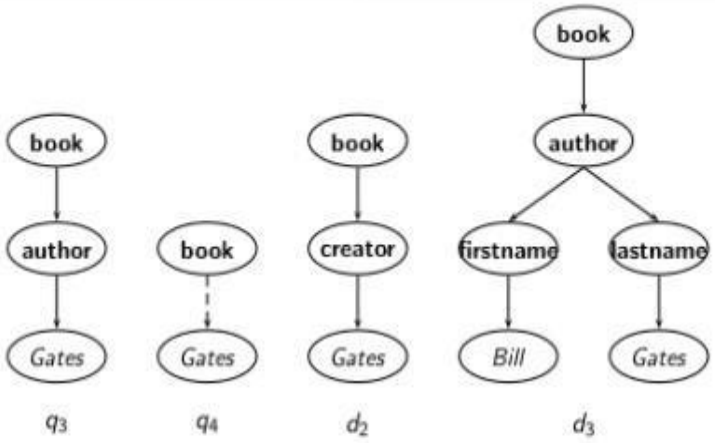
c_q matches c_d iff we can transform c_q into c_d by inserting additional nodes.

Context resemblance example



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

$\mathrm{CR}(c_q, c_d)$ = 3/4 = 0.75. The value of $\mathrm{CR}(c_q, c_d)$ is 1.0 if q and d are identical.



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

$\mathrm{CR}(c_q, c_d)$ = ? $\mathrm{CR}(c_q, c_d)$ = 3/5 = 0.6.