

CSE 3024
WEB MINING

Winter semester 2021-22

Course Objectives

- To focus on a detailed overview of the web mining process and its techniques
- To Understand the basics of Web search with special emphasis on web Crawling
- To understand the basic of indexing and the various type of query processing approaches.
- To appreciate the use of machine learning approaches for Web Content Mining
- To understand the role of hyper links in web structure mining
- To appreciate the various aspects of web usage mining

Course Outcome

- Build a sample search engine using available open-source tools
- Describe the browser security model in web security
- Identify the different components of a web page that can be used for mining
- Apply machine learning concepts to web content mining
- Implement Page Ranking algorithm and modify the algorithm for mining information
- Design a system to harvest information available on the web to build recommender systems
- Analyse social media data using appropriate data/web mining techniques
- Modify an existing search engine to make it personalized

SYLLABUS

- 1. INTRODUCTION TO WEB**
 - 2. WEB CRAWLING**
 - 3. INDEXING**
 - 4. WEB STRUCTURE MINING**
 - 5. WEB CONTENT MINING**
 - 6. WEB USAGE MINING**
 - 7. QUERY PROCESSING**
 - 8. RECENT TRENDS**
-

SYLLABUS

➤ **WEB CRAWLING**

- ❖ A web crawler (also known as a web spider or web robot) is a program or automated script which browses the World Wide Web in a methodical, automated manner.

➤ **WEB INDEXING**

- ❖ **Web indexing** (or Internet **indexing**) refers to various methods for **indexing** the contents of a **website** or of the Internet as a whole.

SYLLABUS

➤ **WEB STRUCTURE MINING**

- ❖ Web structure mining tries to discover useful knowledge from the structure of hyperlinks.

➤ **WEB CONTENT MINING**

- ❖ Web content mining aims to extract/mine useful information or knowledge from web page contents.

➤ **WEB USAGE MINING**

- ❖ Web usage mining refers to the discovery of user access patterns from Web usage logs.

Assessment Methods

Assessment type	Max. Marks	Weightage
Quiz 1	10	10
Quiz 2	10	10
Assignment	10	10
CAT – I	30	15
CAT – II	30	15
FAT	100	40

What is Web Mining?

- an application of data mining techniques to find information patterns from the web data.

Data Mining

What is Data Mining?

- Knowledge discovery in databases (KDD).
- process of discovering useful patterns or knowledge from data sources

Data mining tasks

- Supervised learning (or classification)
- Unsupervised learning (or clustering)
- Association rule mining, and
- Sequential pattern mining

Data mining steps

- Pre-processing
- Data mining
- Post-processing

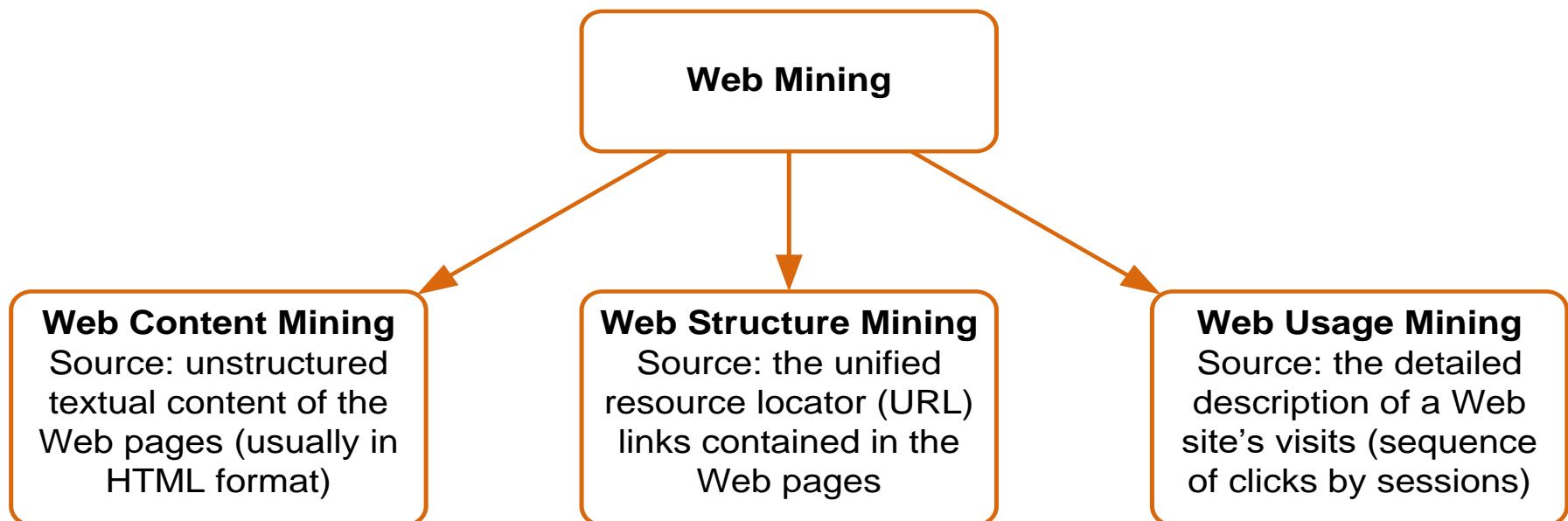
WEB MINING

- Web mining is the use of data mining techniques to automatically discover and extract information from Web documents and services.
- There are three general classes of information that can be discovered by web mining:

Web Graph

Web Content

Web Usage.



WEB MINING

Some of the valuable end users for web mining results are:

- ❖ Search (**Biggest Web Miner by far**)
- ❖ Business intelligence
- ❖ Competitive intelligence
- ❖ Pricing analysis
- ❖ Events
- ❖ Product data
- ❖ Popularity
- ❖ Reputation

Module – 1

INTRODUCTION TO WWW

- 1. Architecture of WWW**
- 2. Web Search Engine**
- 3. Web Security**
- 4. HTTP, HTTPS, URL**
- 5. JAVA and HTML**

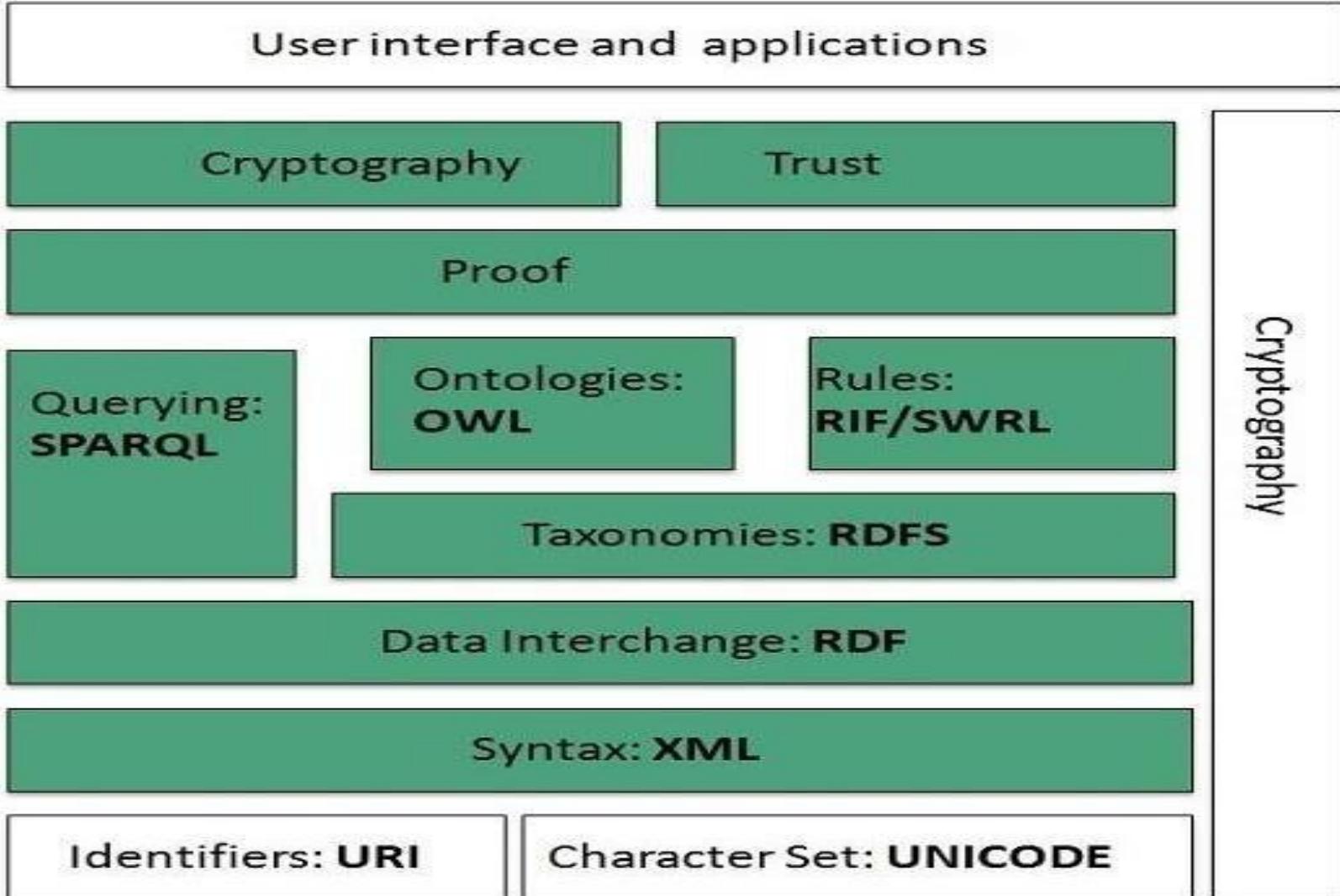
WWW

- ❖ What is WWW?
- ❖ WWW vs. Internet
- ❖ Web Language
 - ❖ HTML
- ❖ Web Protocols
 - ❖ HTTP / FTP
- ❖ URL
 - ❖ HTTP / HTTPS
 - ❖ FTP / MAILTO / USENET
 - ❖ URI / URL / URN
- ❖ W3C
 - ❖ Web Standards / Web Accessibility Standards

BASIC COMPONENTS OF THE WEB

- Web Servers
- Servers
- Web Clients
- HTTP Protocols
- Browser

WEB ARCHITECTURE



WEB ARCHITECTURE

- **Uniform Resource Identifier (URI)** is used to uniquely identify resources on the web and **UNICODE** makes it possible to built web pages that can be read and write in human languages.
 - **XML (Extensible Markup Language)** helps to define common syntax in semantic web.
 - **Resource Description Framework (RDF)** framework helps in defining core representation of data for web. RDF represents data about resource in graph form.
-

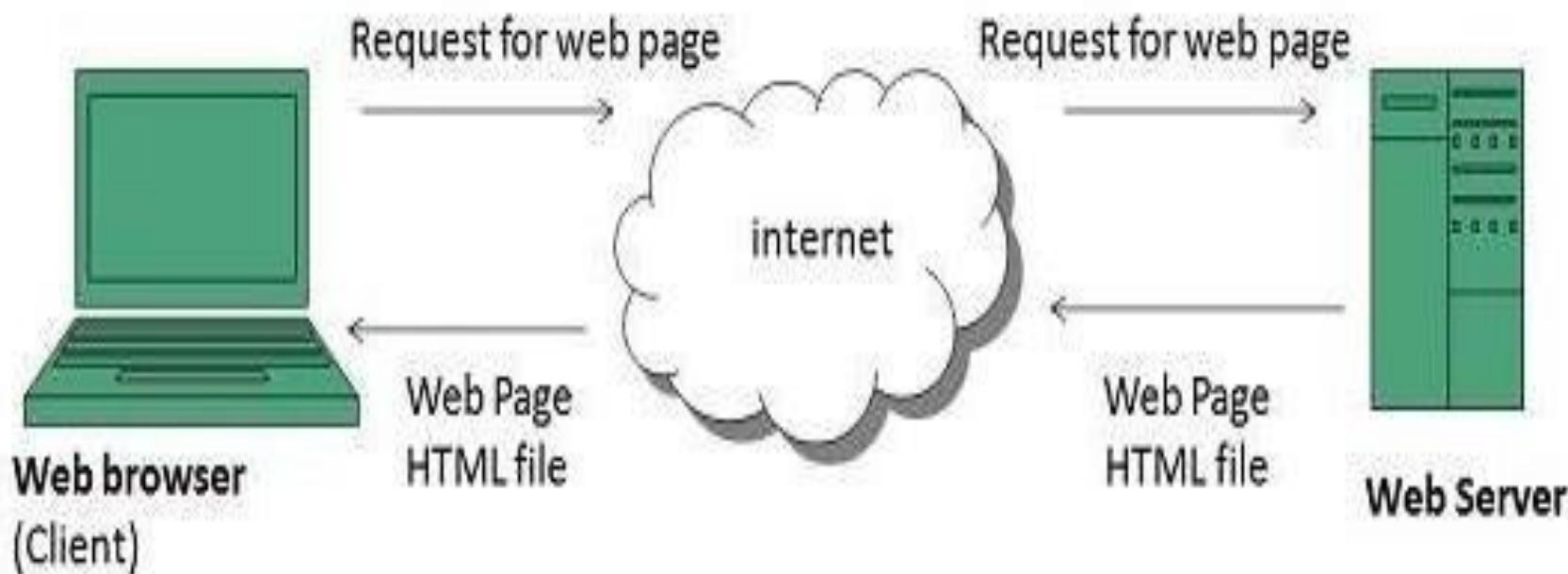
WEB ARCHITECTURE

- **RDF Schema (RDFS)** allows more standardized description of **taxonomies** and other **ontological** constructs.
- **Web Ontology Language (OWL)** offers more constructs over RDFS. It comes in following three versions:
 - OWL Lite for taxonomies and simple constraints.
 - OWL DL for full description logic support.
 - OWL for more syntactic freedom of RDF

WEB ARCHITECTURE

- **RIF** and **SWRL** offers rules beyond the constructs that are available from **RDFs** and **OWL**. Simple Protocol and **RDF Query Language (SPARQL)** is SQL like language used for querying RDF data and OWL Ontologies.
 - All semantic and rules that are executed at layers below **Proof** and their result will be used to prove deductions.
 - **Cryptography** means such as digital signature for verification of the origin of sources is used. On the top of layer **User interface and Applications** layer is built for user interaction.
-

WEB OPERATION



WEB CHALLENGES

- ❖ As originally proposed by Tim Berners-Lee, the Web was intended to improve the management of general information about accelerators and experiments at CERN.

 - ❖ His suggestion was to organize the information used at that institution in a graph-like structure where the nodes are documents describing objects, such as notes, articles, departments, or persons, and the links are relations among them, such as “depends on,” “is part of,” “refers to,” or “uses.”
-

WEB CHALLENGES

- ❖ The framework proposed by Berners-Lee was very general and would work very well for any set of documents, providing flexibility and convenience in accessing large amounts of text.
- ❖ A very important development of this idea was that the documents need not be stored at the same computer or database but rather, could be distributed over a network of computers.
- ❖ Luckily, the infrastructure for this type of distribution, the Internet, had already been developed.

In short, this is how the Web was born.

TimBL Proposed Web vs. Today's Web

- ❖ The recent Web is huge and grows incredibly fast.
 - ❖ About 10 years after the Berners-Lee proposal, the Web was estimated to have 150 million nodes (pages) and 1.7 billion edges (links).
 - ❖ The formal semantics of the Web is very restricted.
 - ❖ nodes are simply web pages and links are of a single type (e.g., “refer to”). The meaning of the nodes and links is not a part of the web system; rather, it is left to web page developers to describe in the page content what their web documents mean and what types of relations they have with the documents to which they are linked.
-

WEB CHALLENGES

- ❖ The Web is the **largest repository** of knowledge in the world, so everyone is tempted to use it. But the big question is **how to find** it. Answering this question has been the basic driving force in developing **web search technologies** (Web Search Engines).
- ❖ The next Challenge is to bring back the **semantics of hypertext documents** (something that was a part of the original web proposal of Berners-Lee) so that we can easily use the vast amount of information available.

TURN WEB DATA INTO WEB KNOWLEDGE

WEB CHALLENGES

- ❖ There are several ways to achieve this:
 - ❖ Use the existing Web and apply **sophisticated search techniques.**
 - ❖ Change the way in which we create web pages.

WEB SEARCH ENGINE

A web search engine is a coordinated set of programs that includes:

- ❖ A spider (also called a "crawler" or a "bot") that goes to every page or representative pages on every Web site that wants to be searchable and reads it, using hypertext links on each page to discover and read a site's other pages.
 - ❖ A program that creates a huge index (sometimes called a "**catalog**") from the pages that have been read.
 - ❖ A program that receives your search request, compares it to the entries in the index, and returns results to you.
-

WEB SEARCH ENGINE

List of Top 10 Most popular search engine:

- Google
 - Bing
 - Yahoo
 - Ask
 - AOL
 - Baidu
 - Wolframalpha
 - DuckDuckGo
 - Internet Archive
 - Chacha.com
-

WEB SEARCH ENGINE

- ❖ Web search engines explore the existing (semantics-free) structure of the Web and try to find documents that match user search criteria.

 - ❖ The basic idea is to use a set of words (or terms) that the user specifies and retrieve documents that include (or do not include) those words. – **Keyword Search**

 - ❖ Further IR techniques are used to avoid terms that are too general and too specific and to take into account term distribution throughout the entire **body of documents** as well as to **explore document similarity**.
-

WEB SEARCH ENGINE

- ❖ Natural language processing approaches are also used to analyze term context or lexical information, or to combine several terms into phrases.
 - ❖ After retrieving a set of documents ranked by their degree of matching the keyword query, they are further ranked by importance (popularity, authority), usually based on the web link structure.
-

TOPIC DIRECTORIES

- Web pages are organized into hierarchical structures that reflect their meaning. These are known as *topic directories*, or simply *directories*, and are available from almost all web search portals.

 - The largest is being developed under the Open Directory Project (dmoz.org) and is used by Google in their Web Directory.

 - The directory structure is often used in the process of web search to better match user criteria or to specialize a search within a specific set of pages from a given category.
-

TOPIC DIRECTORIES

- The directories are usually created manually with the help of thousands of web page creators and editors.

 - There are also approaches to do this automatically by applying machine learning methods for classification and clustering.
-

SEMANTIC WEB

- *Semantic web* is a initiative led by the web consortium (w3c.org).
- Its main objective is to bring formal knowledge representation techniques into the Web.
- It is widely acknowledged that the Web is like a “**fancy fax machine**” used to send good-looking documents worldwide.
 - The problem here is that the nice format of web pages is very difficult for computers to understand—something that we expect search engines to do.

SEMANTIC WEB

- The main idea behind the semantic web is to add formal descriptive material to each web page that although invisible to people would make its content easily understandable by computers.

- Thus, the Web would be organized and turned into the largest knowledge base in the world, which with the help of advanced reasoning techniques developed in the area of artificial intelligence would be able not just to provide ranked documents that match a keyword search query, but would also be able to answer questions and give explanations.

<http://www.w3.org/2001/sw/>

WEB SEARCH ENGINE CHALLENGES

- Spam
- Content Quality
- Quality Evaluation
- Web Convention
- Duplicate Hosts
- Vaguely-Structured Data

Source: Henzinger, Monika R., Rajeev Motwani, and Craig Silverstein. "Challenges in web search engines." *ACM SIGIR Forum*. Vol. 36. No. 2. ACM, 2002.

SEARCH ENGINE SPAM

- [Silverstein et al., 1999] showed that for 85% of the queries only the first result screen is requested.
 - Thus, inclusion in the first result screen, which usually shows the top 10 results, can lead to an increase in traffic to a web site, while exclusion means that only a small fraction of the users will actually see a link to the web site.
 - The result of this process is commonly called *search engine spam*. To achieve high rankings, authors use either of these following methods
 - a text-based approach
 - a link-based approach
 - a cloaking approach (spamdexing technique)
 - a combination thereof.
-

SEARCH ENGINE SPAM

- [Silverstein et al., 1999] showed that for 85% of the queries only the first result screen is requested.
 - Thus, inclusion in the first result screen, which usually shows the top 10 results, can lead to an increase in traffic to a web site, while exclusion means that only a small fraction of the users will actually see a link to the web site.
 - The result of this process is commonly called *search engine spam*. To achieve high rankings, authors use either of these following methods
 - a text-based approach
 - a link-based approach
 - a cloaking approach (spamdexing technique)
 - a combination thereof.
-

SEARCH ENGINE SPAM

- Unfortunately, spamming has become so prevalent that every commercial search engine has had to take measures to identify and remove spam. Without such measures, the quality of the rankings suffers severely.

 - One approach to deal with the spam problem is to construct a spam classifier that tries to label pages as spam or not-spam
-

CONTENT QUALITY

- The web is full of noisy, low-quality, unreliable, and indeed contradictory content.
 - A reasonable approach for relatively high-quality content would be to assume that every document in a collection is authoritative and accurate, design techniques for this context, and then tweak the techniques to incorporate the possibility of low-quality content.
 - In designing a high-quality search engine, one has to start with the assumption that a typical document cannot be "trusted" in isolation; rather, it is the synthesis of a large number of low-quality documents that provides the best set of results.
-

CONTENT QUALITY

- There have been link-based approaches, for instance PageRank [Brin and Page, 1998], for estimating the quality of web pages. However, PageRank only uses the link structure of the web to estimate page quality.

 - It seems to us that a better estimate of the quality of a page requires additional sources of information, both within a page (e.g., the reading level of a page) and across different pages (e.g., correlation of content).
-

QUALITY EVALUATION

- Evaluating the quality of different ranking algorithms is a notoriously difficult problem.
 - Commercial search engines have the benefit of large amounts of user-behaviour data they can use to help evaluate ranking.
 - Users usually will not make the effort to give explicit feedback but nonetheless leave implicit feedback information such as the results on which they clicked.
 - The research issue is to exploit the implicit feedback to evaluate different ranking strategies.
-

WEB CONVENTIONS

- For Example, many webmasters use the anchor text of a link to provide a succinct description of the target page.
 - Since most authors behave this way, we will refer to these rules as web conventions, even though there has been no formalization or standardization of such rules.
 - Search engines rely on these web conventions to improve the quality of their results. Consequently, when webmasters violate these conventions they can confuse search engines.
 - The main issue here is to identify the various conventions that have evolved organically and to develop techniques for accurately determining when the conventions are being violated.
-

DUPLICATE HOSTS

- Web search engines try to avoid crawling and indexing duplicate and near-duplicate pages, as they do not add new information to the search results and clutter up the results.

 - However, if a search engine can avoid crawling the duplicate content in the first place, the gain is even larger.

 - In general, predicting whether a page will end up being a duplicate of an already-crawled page is chancy work, but the problem becomes more tractable if we limit it to finding duplicate hosts, that is, two hostnames that serve the same content.
-

VAGUELY-STRUCTURED DATA

- The degree of structure present in data has had a strong influence on techniques used for search and retrieval.
 - At one extreme, the database community has focused on highly-structured, relational data, while at the other the information retrieval community has been more concerned with essentially unstructured text documents.
 - Of late, there has been some movement toward the middle with the database literature considering the imposition of structure over almost-structured data.
-

VAGUELY-STRUCTURED DATA

- In a similar vein, document management systems use accumulated meta-information to introduce more structure.
 - The emergence of XML has led to a flurry of research involving extraction, imposition, or maintenance of partially-structured data.
-

SEMANTIC WEB vs. SEMANTIC SEARCH

- **The Semantic Web** is a set of technologies for representing, storing, and querying information. Although these technologies can be used to store textual data, they typically are used to store smaller bits of data.
 - **Semantic search** is the process of typing something into a search engine and getting more results than just those that feature the exact keyword you typed into the search box. Semantic search will take into account the context and meaning of your search terms. It's about understanding the assumptions that the searcher is making when typing in that search query.
-

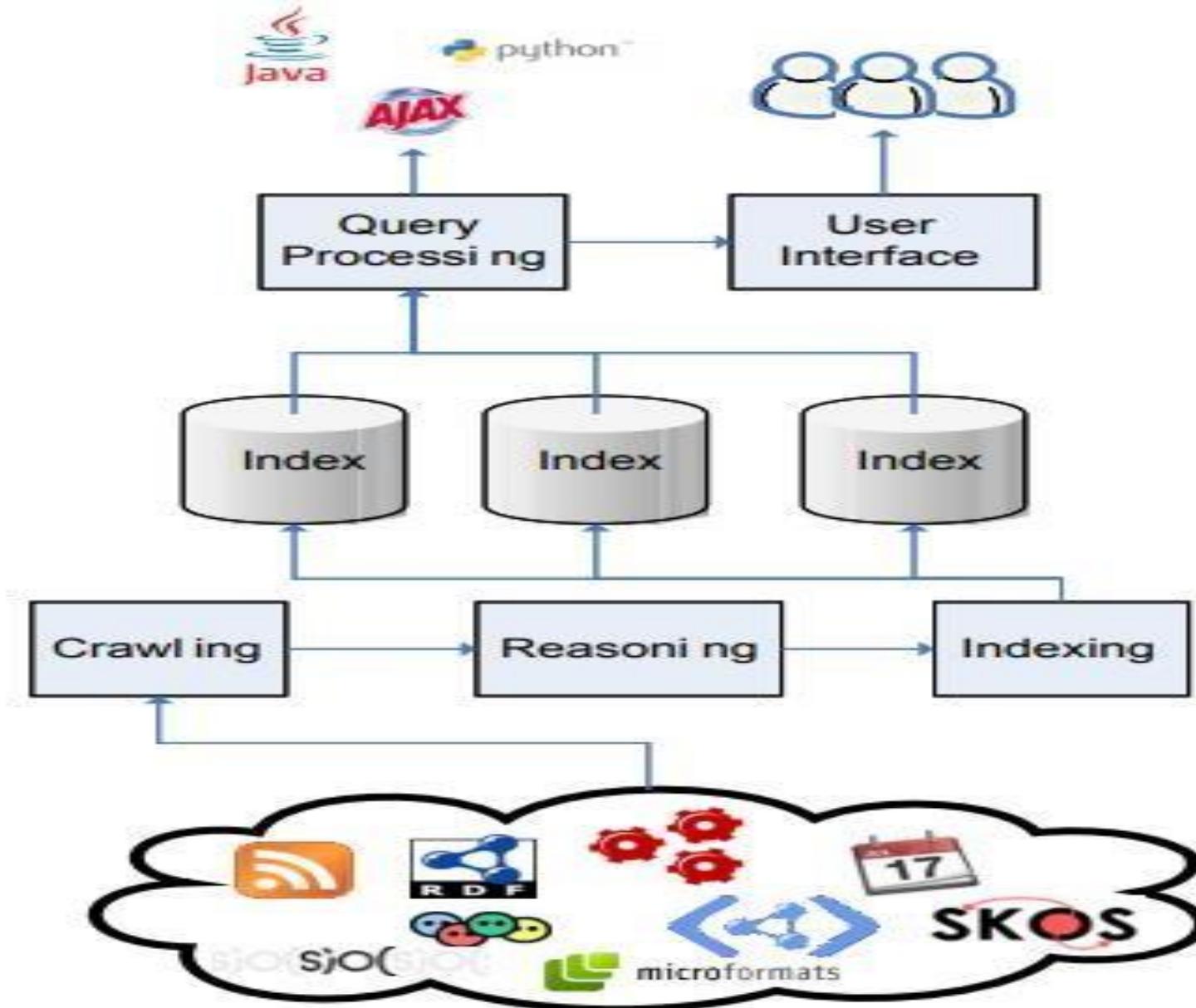
SEMANTIC WEB vs. SEMANTIC SEARCH

- Semantic search focuses on the text, but the semantic web focuses on pulling data from multiple sources and multiple formats.
 - For example, if you type in the word “Blackhawks” into your search bar you don’t just want to get listings that have the word “Blackhawks” in them. A semantic search will return listings about the Native American tribe as well as the Chicago hockey. You will also get supporting terms like “hockey lessons” and “Stanley Cup,” even if they never mention anything about the Blackhawks exactly.
-

SEMANTIC WEB SEARCH ENGINE CHALLENGES

- The architecture of a semantic search engine must scale to the Web.
 - Dealing with data rather than documents requires a different indexing approach compared to traditional information retrieval systems.
 - Data from the Web is of varying quality, which poses challenges for data cleansing and entity consolidation.
 - The schema of the data is not known a priori, which makes building user interfaces difficult.
-

SEMANTIC WEB SEARCH ENGINE ARCHITECTURE



WEB CRAWLERS

- ❖ The approach is to have web pages organized by topic or to search a collection of pages indexed by keywords.
 - ❖ The former is done by topic directories and the latter, by search engines.
- ❖ Collecting “all” web documents can be done by browsing the Web systematically and exhaustively and storing all visited pages. This is done by *crawlers* (also called *spiders* or *robots*).
- ❖ Ideally, all web pages are linked (there are no unconnected parts of the web graph) and there are no multiple links and nodes. Then the job of a crawler is simple:
 - ❖ to run a complete *graph search algorithm*, such as *depth-first* or *breadth-first* search, and store all visited pages.
- ❖ A good example of such a crawler is **WebSPHINX**.
(<https://www.cs.cmu.edu/~rcm/websphinx/>)

INDEXING AND KEYWORD SEARCH

- ❖ Generally, there are two types of data: structured and unstructured.
 - ❖ *Structured data* have keys (attributes, features) associated with each data item that reflect its content, meaning, or usage.
 - ❖ A typical example of structured data is a relational table in a database.
 - ❖ The problem with using structured data is the cost associated with the process of structuring them. The information that people use is available primarily in unstructured form.
 - ❖ This is the *keyword search* approach known from the area of *information retrieval* (IR).
 - ❖ The idea of IR is to retrieve documents by using a simple Boolean criterion: the presence or absence of specific words (keywords, terms) in the documents
-

INDEXING AND KEYWORD SEARCH

File Edit View Favorites Tools Help

Back Favorites Search Address http://www.artsci.ccsu.edu/Departments.htm Go

CCSU Central Connecticut State University  Start with a Dream. Finish with a Future.

The School of Arts & Sciences

Departments

[Department Chairs](#), [Locations](#), [Phone Numbers](#)

<u>Anthropology</u>	<u>History</u>
<u>Art</u>	<u>Mathematical Sciences</u>
<u>Biological Sciences</u>	<u>Modern Languages</u>
<u>Chemistry</u>	<u>Music</u>
<u>Communication</u>	<u>Philosophy</u>
<u>Computer Science</u>	<u>Physics/Earth Sciences</u>
<u>Criminal Justice</u>	<u>Political Science</u>
<u>Design</u>	<u>Psychology</u>
<u>Economics</u>	<u>Sociology</u>
<u>English</u>	<u>Theatre</u>
<u>Geography</u>	

[[A&S Home](#)] [[A-Z Directory](#)] [[Departments](#)] [[About the School](#)]

page last updated: 10/27/04

Internet

INDEXING AND KEYWORD SEARCH

A screenshot of a Microsoft Internet Explorer window displaying the website for the Music department at Central Connecticut State University. The title bar reads "Music - Microsoft Internet Explorer". The address bar shows the URL "http://www.artsci.ccsu.edu/Departments/Music.html". The page content includes the university's logo, the School of Arts and Sciences name, and a section titled "Music" with a descriptive paragraph about the department's programs and facilities. It also lists the department chair and contact information, along with a link to the department website.

The School of Arts and Sciences
Central Connecticut State University

Music

Students majoring in music may pursue either a BS in Music education degree, the professional degree that certifies them to teach music in the public schools, or a BA in music, with specializations in either performance, music history, theory/composition, or jazz studies. Full-time and associate faculty are active in the United States and abroad performing, conducting, and presenting scholarly papers. The department's computer lab is equipped with MIDI keyboards and the industry's leading music software. The Music Department is the New England center for Orff Schulwerk training and the host for Connecticut's middle school/high school music festival and the Summer Music Institute, a national in-service program for music educators.

PROGRAMS OF STUDY: BS, BA, MS

DEPARTMENT CHAIR
Pamela Perry

Location: Welte Hall 101
Phone: 832-2900

[Department Website](#)

INDEXING AND KEYWORD SEARCH

- ❖ The first step is to fetch the documents from the Web, remove the HTML tags, and store the documents as plain text files.
 - ❖ Then the keyword search approach can be used to answer such queries as:
 - ❖ Find documents that contain the word *computer* and the word *programming*.
 - ❖ Find documents that contain the word *program*, but not the word *programming*.
 - ❖ Find documents where the words *computer* and *lab* are adjacent. This query is called *proximity query*, because it takes into account the lexical distance between words. Another way to do it is by searching for the phrase *computer lab*.
-

WEB DOCUMENT REPRESENTATION

- ❖ Documents are *tokenized*; that is, all punctuation marks are removed and the character strings without spaces are considered as tokens (words, also called *terms*).
 - ❖ All characters in the documents and in the query are converted to upper or lower case.
 - ❖ Words are reduced to their canonical form (stem, base, or root).
 - ❖ For example, variant forms such as *is* and *are* are replaced with *be*, various endings are removed, or the words are transformed into their root form, such as *programs* and *programming* into *program*.
-

WEB DOCUMENT REPRESENTATION

- ❖ This process, called *stemming*, uses morphological information to allow matching different variants of words.

 - ❖ Articles, prepositions, and other common words that appear frequently in text documents but do not bring any meaning or help distinguish documents are called *stopwords*. These words are usually removed.
 - ❖ Examples are *a*, *an*, *the*, *on*, *in*, and *at*.

 - ❖ The collection of words that are left in the document after all those steps is different from the original document and may be considered as a *formal representation* of the document.
-

WEB DOCUMENT REPRESENTATION

Document ID	Document Name	Words	Terms
d_1	Anthropology	114	86
d_2	Art	153	105
d_3	Biology	123	91
d_4	Chemistry	87	58
d_5	Communication	124	88
d_6	Computer Science	101	77
d_7	Criminal Justice	85	60
d_8	Economics	107	76
d_9	English	116	80
d_{10}	Geography	95	68
d_{11}	History	108	78
d_{12}	Mathematics	89	66
d_{13}	Modern Languages	110	75
d_{14}	Music	137	91
d_{15}	Philosophy	85	54
d_{16}	Physics	130	100
d_{17}	Political Science	120	86
d_{18}	Psychology	96	60
d_{19}	Sociology	99	66
d_{20}	Theatre	116	80
Total number of words/terms		2195	1545
Number of different words/terms		744	671

WEB DOCUMENT REPRESENTATION

- ❖ The words are counted after tokenizing the plain text versions of the documents (without the HTML structures). The term counts are taken after removing the stopwords but without stemming.
 - ❖ To emphasize this difference, we call the words in this collection *terms*. The collection of words (terms) in the entire set of documents is called the *text corpus*.
 - ❖ The terms that occur in a document are in fact the *parameters* (also called *features*, *attributes*, or *variables* in different contexts) of the document representation.
-

TYPE OF WEB DOCUMENT REPRESENTATION

- ❖ The simplest way to use a term as a feature in a document representation is to check whether or not the term occurs in the document. Thus, the term is considered as a Boolean attribute, so the representation is called *Boolean*.
 - ❖ The value of a term as a feature in a document representation may be the number of occurrences of the term (*term frequency*) in the document or in the entire corpus.
 - ❖ Document representation that includes the term frequencies but not the term positions is called a *bag-of-words representation* because formally it is a multiset or bag.
-

WEB DOCUMENT REPRESENTATION

- ❖ Term positions may be included along with the frequency. This is a “**complete**” representation that preserves most of the information and may be used to generate the original document from its representation.

- ❖ The purpose of the document representation is to help the process of keyword matching.
 - ❖ However, it may also result in loss of information, which generally increases the number of documents in response to the keyword query. Thus some irrelevant documents may also be returned.

WEB DOCUMENT REPRESENTATION

Document ID	<i>lab</i>	<i>laboratory</i>	<i>programming</i>	<i>computer</i>	<i>program</i>
d_1	0	0	0	0	1
d_2	0	0	0	0	1
d_3	0	1	0	1	0
d_4	0	0	0	1	1
d_5	0	0	0	0	0
d_6	0	0	1	1	1
d_7	0	0	0	0	1
d_8	0	0	0	0	1
d_9	0	0	0	0	0
d_{10}	0	0	0	0	0
d_{11}	0	0	0	0	0
d_{12}	0	0	0	1	0
d_{13}	0	0	0	0	0
d_{14}	1	0	0	1	1
d_{15}	0	0	0	0	1
d_{16}	0	0	0	0	1
d_{17}	0	0	0	0	1
d_{18}	0	0	0	0	0
d_{19}	0	0	0	0	1
d_{20}	0	0	0	0	0

WEB DOCUMENT REPRESENTATION

Document ID	<i>lab</i>	<i>laboratory</i>	<i>programming</i>	<i>computer</i>	<i>program</i>
d_1	0	0	0	0	[71]
d_2	0	0	0	0	[7]
d_3	0	[65,69]	0	[68]	0
d_4	0	0	0	[26]	[30,43]
d_5	0	0	0	0	0
d_6	0	0	[40,42]	[1,3,7,13,26,34]	[11,18,61]
d_7	0	0	0	0	[9,42]
d_8	0	0	0	0	[57]
d_9	0	0	0	0	0
d_{10}	0	0	0	0	0
d_{11}	0	0	0	0	0
d_{12}	0	0	0	[17]	0
d_{13}	0	0	0	0	0
d_{14}	[42]	0	0	[41]	[71]
d_{15}	0	0	0	0	[37,38]
d_{16}	0	0	0	0	[81]
d_{17}	0	0	0	0	[68]
d_{18}	0	0	0	0	0
d_{19}	0	0	0	0	[51]
d_{20}	0	0	0	0	0

WEB DOCUMENT REPRESENTATION

- ❖ For example, stemming of *programming* would change the second query and allow the first one to return more documents (its original purpose is to identify the Computer Science department, but stemming would allow more documents to be returned, as they all include the word *program* or *programs* in the sense of “program of study”).
 - ❖ Therefore, stemming should be applied with care and even avoided, especially for Web searches, where a lot of common words are used with specific technical meaning.
-

WEB DOCUMENT REPRESENTATION

- ❖ This problem is also related to the issue of context (lexical or semantic), which is generally lost in keyword search.
 - ❖ A partial solution to the latter problem is the use of proximity information or lexical context.
 - ❖ Partial solution to the latter problem is the use of proximity information or lexical context.
 - ❖ For this purpose a richer document representation can be used that preserves term positions.
 - ❖ Some punctuation marks can be replaced by placeholders (tokens that are left in a document but cannot be used for searching), so that part of the lexical structure of the document, such as sentence boundaries, can be preserved.
-

WEB DOCUMENT REPRESENTATION

- ❖ Another approach, called *part-of-speech tagging*, is to attach to words tags that reflect their part-of-speech roles (e.g., verb or noun).
 - ❖ For example, the word *can* usually appears in the stopword list, but as a noun it may be important for a query.

WEB SECURITY

GOTO WEB SECURITY



WEB APPLICATION SECURITY

“This Site is Secure”

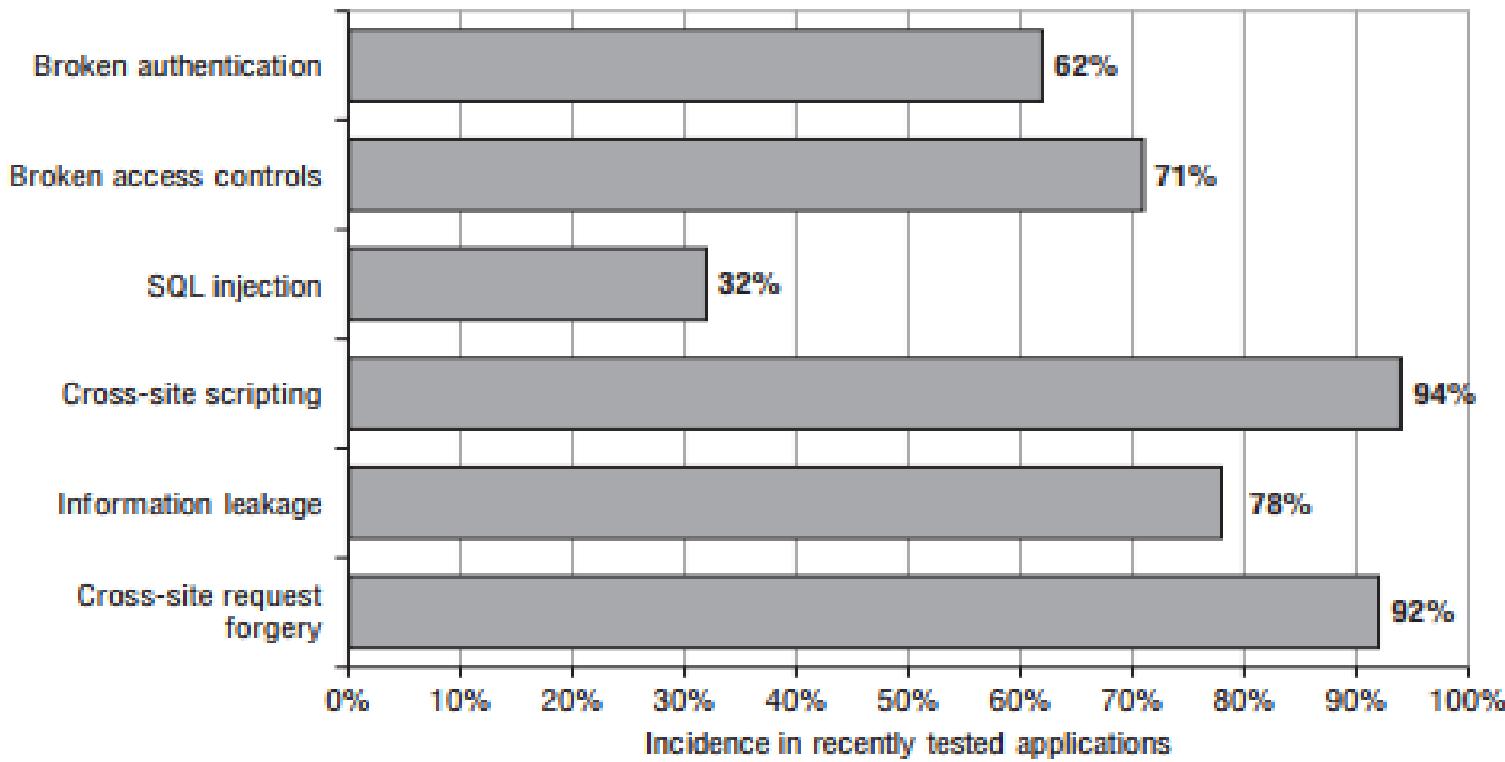
- Most applications state that they are secure because they use SSL.
For example:

This site is absolutely secure. It has been designed to use 128-bit Secure Socket Layer (SSL) technology to prevent unauthorized users from viewing any of your information. You may use this site with peace of mind that your data is safe with us.

- Increasingly, organizations also cite their compliance with Payment Card Industry (PCI) standards to reassure users that they are secure.
-

WEB APPLICATION SECURITY

- In fact, the majority of web applications are insecure, despite the widespread usage of SSL technology and the adoption of regular PCI scanning.



WEB APPLICATION SECURITY

- **Broken authentication (62%)** — This category of vulnerability encompasses various defects within the application's login mechanism, which may enable an attacker to guess weak passwords, launch a brute-force attack, or bypass the login.
 - **Broken access controls (71%)** — This involves cases where the application fails to properly protect access to its data and functionality, potentially enabling an attacker to view other users' sensitive data held on the server or carry out privileged actions.
-

WEB APPLICATION SECURITY

- **SQL injection (32%)** — This vulnerability enables an attacker to submit crafted input to interfere with the application's interaction with back-end databases. An attacker may be able to retrieve arbitrary data from the application, interfere with its logic, or execute commands on the database server itself.
 - **Cross-site scripting (94%)** — This vulnerability enables an attacker to target other users of the application, potentially gaining access to their data, performing unauthorized actions on their behalf, or carrying out other attacks against them.
-

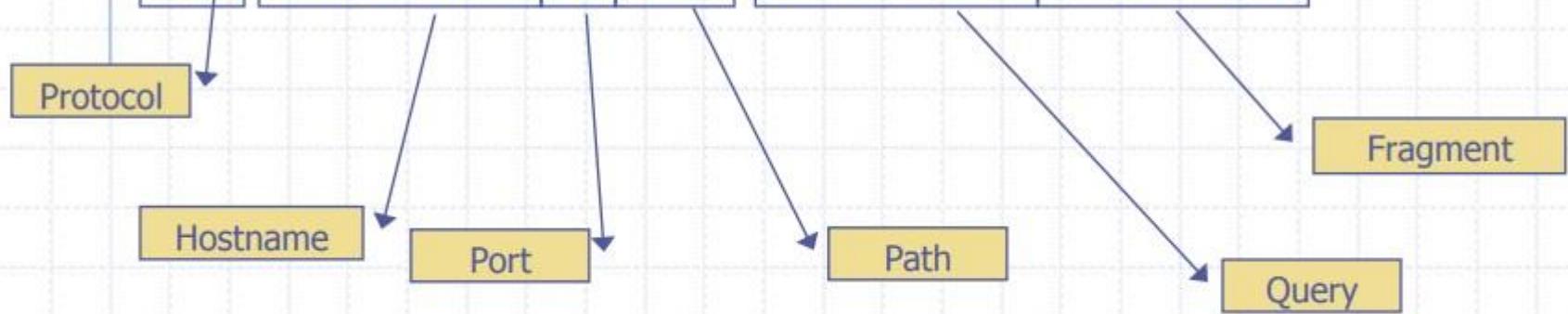
WEB APPLICATION SECURITY

- **Information leakage (78%)** — This involves cases where an application divulges sensitive information that is of use to an attacker in developing an assault against the application, through defective error handling or other behaviour.
 - **Cross-site request forgery (92%)** — This flaw means that application users can be induced to perform unintended actions on the application within their user context and privilege level. The vulnerability allows a malicious web site visited by the victim user to interact with the application to perform actions that the user did not intend.
-

WEB SECURITY MODEL

◆ Example:

http://stanford.edu:81/class?name=cs155#homework



- ◆ Special characters are encoded as hex:
 - %0A = newline
 - %20 or + = space, %2B = + (special exception)

- HTML DOM
- SAME ORIGIN POLICY

SAME ORIGIN POLICY

Website origin = (scheme, domain, port)

Compared URL	Outcome	Reason
<code>http://www.example.com/dir/page.html</code>	Success	Same protocol and host
<code>http://www.example.com/dir2/other.html</code>	Success	Same protocol and host
<code>http://www.example.com:81/dir/other.html</code>	Failure	Same protocol and host but different port
<code>https://www.example.com/dir/other.html</code>	Failure	Different protocol
<code>http://en.example.com/dir/other.html</code>	Failure	Different host
<code>http://example.com/dir/other.html</code>	Failure	Different host (exact match required)
<code>http://v2.www.example.com/dir/other.html</code>	Failure	Different host (exact match required)

RELAXING SAME ORIGIN POLICY

- `document.domain` property
 - For example, cooperating scripts in documents loaded from `orders.example.com` and `catalog.example.com` might set their `document.domain` properties to “`example.com`”, thereby making the documents appear to have the same origin and enabling each document to read properties of the other.
- Cross-Origin Resource Sharing
 - It allows servers to use a header to explicitly list origins that may request a file or to use a wildcard and allow a file to be requested by any site

RELAXING SAME ORIGIN POLICY

- Cross-document messaging
 - Calling the postMessage() method on a Window object asynchronously fires an "onmessage" event in that window, triggering any user-defined event handlers.
- WebSockets
 - they recognize when a WebSocket URI is used, and insert an **Origin:** header into the request that indicates the origin of the script requesting the connection.

WEB APPLICATION HACKER'S METHODOLOGY

General Guidelines

- Remember that several characters have special meaning in different parts of the HTTP request. When you are modifying the data within requests, you should URL-encode these characters to ensure that they are interpreted in the way you intend.

 - Furthermore, note that entering URL-encoded data into a form usually causes your browser to perform another layer of encoding. For example, submitting %00 in a form will probably result in a value of %2500 being sent to the server. For this reason it is normally best to observe the final request within an intercepting proxy.
-

WEB APPLICATION HACKER'S METHODOLOGY

General Guidelines

- Many tests for common web application vulnerabilities involve sending various crafted input strings and monitoring the application's responses for anomalies, which indicate that a vulnerability is present.
 - In any case where specific crafted input results in behaviour associated with a vulnerability (such as a particular error message), you should double-check whether submitting benign input in the relevant parameter also causes the same behaviour. If it does, your tentative finding is probably a false positive.
-

WEB APPLICATION HACKER'S METHODOLOGY

General Guidelines

- Applications typically accumulate an amount of state from previous requests, which affects how they respond to further requests. Sometimes, when you are trying to investigate a tentative vulnerability and isolate the precise cause of a particular piece of anomalous behaviour, you must remove the effects of any accumulated state.

WEB APPLICATION HACKER'S METHODOLOGY

General Guidelines

- Some applications use a load-balanced configuration in which consecutive HTTP requests may be handled by different back-end servers at the web, presentation, data, or other tiers.
 - some successful attacks will result in a change in the state of the specific server that handles your requests — such as the creation of a new file within the web root.
 - To isolate the effects of particular actions, it may be necessary to perform several identical requests in succession, testing the result of each until your request is handled by the relevant server.
-

WEB APPLICATION HACKER'S METHODOLOGY

- ❖ Map the Application's Content
 - ❖ Analyse the Application
 - ❖ Test Client-Side Controls
 - ❖ Test the Authentication Mechanism
 - ❖ Test the Session Management Mechanism
 - ❖ Test Access Controls
 - ❖ Test for Input-Based Vulnerabilities
 - ❖ Test for Function-Specific Input Vulnerabilities
 - ❖ Test for Logic Flaws
 - ❖ Test for Shared Hosting Vulnerabilities
 - ❖ Test for Application Server Vulnerabilities
 - ❖ Miscellaneous Checks
 - ❖ Follow Up Any Information Leakage
-

WEB APPLICATION HACKER'S METHODOLOGY

Map the Application's Content

- Explore Visible Content
- Consult Public Resources
- Discover Hidden Content
- Discover Default Content
- Enumerate Identifier-Specified Functions
- Test for Debug Parameters

Analyse the Application

- Identify Functionality
 - Identify Data Entry Points
 - Identify the Technologies Used
 - Map the Attack Surface
-

WEB APPLICATION HACKER'S METHODOLOGY

Test Client-Side Controls

- Test Transmission of Data Via the Client
- Test Client-Side Controls Over User Input
- Test Browser Extension Components

Test the Authentication Mechanism

- Understand the Mechanism
- Test Password Quality
- Test for Username Enumeration
- Test Resilience to Password Guessing
- Test Any Account Recovery Function
- Test Any Remember Me Function
- Test Any Impersonation Function

WEB APPLICATION HACKER'S METHODOLOGY

- Test Username
- Test Predictability of Autogenerated Credentials
- Check for Unsafe Transmission of Credentials
- Check for Unsafe Distribution of Credentials
- Test for Insecure Storage
- Test for Logic Flaws
- Exploit Any Vulnerabilities to Gain Unauthorized Access

Test the Session Management Mechanism

- Understand the Mechanism
 - Test Tokens for Meaning
 - Test Tokens for Predictability
 - Check for Insecure Transmission of Tokens
-

WEB APPLICATION HACKER'S METHODOLOGY

- Check for Disclosure of Tokens in Logs
- Check Mapping of Tokens to Sessions
- Test Session Termination
- Check for Session Fixation
- Check for CSRF
- Check Cookie Scope

Test Access Controls

- Understand the Access Control Requirements
 - Test with Multiple Accounts
 - Test with Limited Access
 - Test for Insecure Access Control Methods
-

WEB APPLICATION HACKER'S METHODOLOGY

Test for Input-Based Vulnerabilities

- Fuzz All Request Parameters
- Test for SQL Injection
- Test for XSS and Other Response Injection
- Test for OS Command Injection
- Test for Path Traversal
- Test for Script Injection
- Test for File Inclusion

Test for Function-Specific Input Vulnerabilities

- Test for SMTP Injection
 - Test for Native Software Vulnerabilities
 - Test for SOAP Injection
-

WEB APPLICATION HACKER'S METHODOLOGY

- Test for LDAP Injection
- Test for XPath Injection
- Test for Back-End Request Injection
- Test for XXE Injection

Test for Logic Flaws

- Identify the Key Attack Surface
- Test Multistage Processes
- Test Handling of Incomplete Input
- Test Trust Boundaries
- Test Transaction Logic

WEB APPLICATION HACKER'S METHODOLOGY

Test for Shared Hosting Vulnerabilities

- Test Segregation in Shared Infrastructures
- Test Segregation Between ASP-Hosted Applications

Test for Application Server Vulnerabilities

- Test for Default Credentials
- Test for Default Content
- Test for Dangerous HTTP Methods
- Test for Proxy Functionality
- Test for Virtual Hosting Misconfiguration
- Test for Web Server Software Bugs
- Test for Web Application Firewalling



WEB APPLICATION HACKER'S METHODOLOGY

Miscellaneous Checks

- Check for DOM-Based Attacks
- Check for Local Privacy Vulnerabilities
- Check for Weak SSL Ciphers
- Check Same-Origin Policy Configuration

WEB CRAWLING

Basic Crawler Algorithm: Breadth-First

POPULAR WEB CRAWLERS IN THE MARKET

Bot Name	% of Sites Crawled	Bot Type
Googlebot	96%	Search Bot
Baidu Spider	89%	Search Bot
MSN Bot/BingBot	82%	Search Bot
Yandex Bot	73%	Search Bot
Soso Spider	61%	Search Bot
ExaBot	35%	Search Bot
Sogou Spider	31%	Search Bot
Google Plus Share	24%	Crawler
Facebook External Hit	24%	Crawler
Google Feedfetcher	22%	Feed Fetcher

Web Crawler?

- The process or program used by search engines to download pages from the web for later processing by a search engine that will index the downloaded pages to provide fast searches
- A program or automated script which browses the World Wide Web in a methodical, automated manner
- Also known as web spiders and web robots
- Less used names- ants, bots and worms

Why web Crawlers?

- ❖ Internet has a wide expanse of Information
- ❖ **Finding relevant information requires an efficient mechanism**
- ❖ **Web Crawlers provide that scope to the search engine**



About 24,70,00,000 results (1.02 seconds)

Ad · viteee.vit.ac.in/ 0416 220 2020

B. Tech Admissions Open In VIT - Online Application Form 2020

Ranked as one of the best engineering college in India for 3 years in a row by NIRF. World standard labs, e-Journals, technology business incubator at VIT. World ranked institution. Advanced teaching method. 300+ International MOU's. Best Placements. 700+ Recruiters. Programmes Offered · View Syllabus · Eligibility Criteria · Exam Pattern · Fees Structure

vit.ac.in ▾

VIT | No.1 Private Institution for Innovation

Established in 1984, VIT is a No.1 Progressive Educational Institution & Top Ranking University in India. Its a dedicated to the pursuit of excellence..

You've visited this page 2 times. Last visit: 27/8/19

VITEEE

Please visit only www.vit.ac.in or
<https://viteee.vit.ac.in> for ...

Admissions

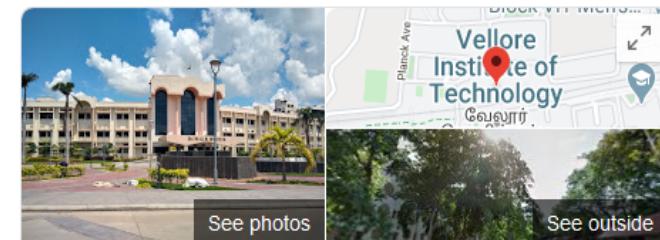
Postgraduate - Undergraduate

VIT | Chennai

Undergraduate - Admissions -
Programmes Offered - SAS - ...

Postgraduate

PG Application - Programmes



Vellore Institute of Technology

[Website](#) [Directions](#) [Save](#)

Private university in Vellore, Tamil Nadu

Vellore Institute of Technology is a private university located in Vellore, Tamil Nadu, India. Founded in 1984, as Vellore Engineering College, by G. Viswanathan, the institution offers 20 undergraduate, 34 postgraduate, four integrated and four research programs. [Wikipedia](#)

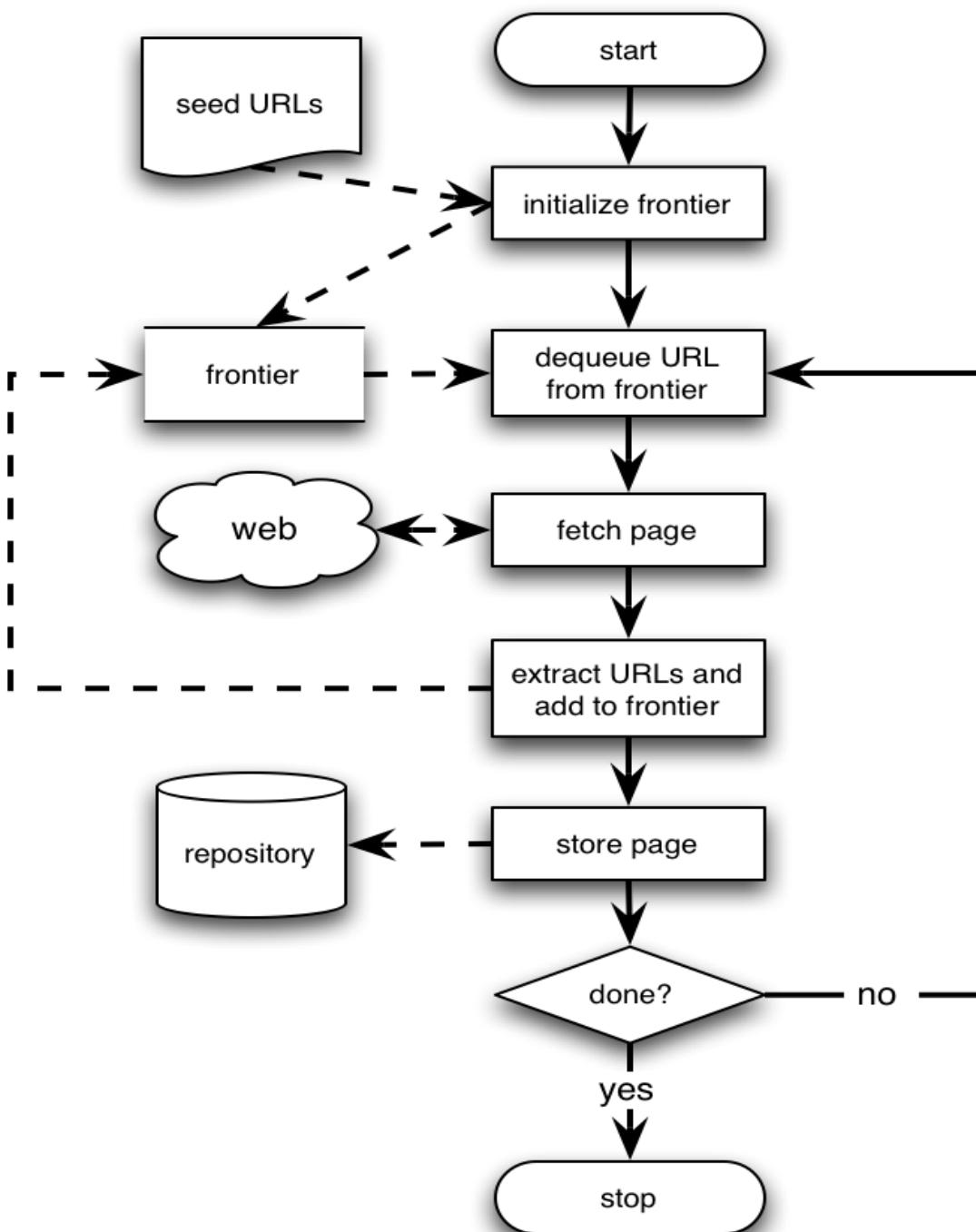
Crawl “all” Web pages?

- **Problem:** No catalog of all accessible URLs on the Web.
- **Solution:**
 - start from a given set of URLs
 - Progressively fetch and scan them for new outlinking URLs
 - fetch these pages in turn
 - Submit the text in page to a text indexing system

How does web crawler work?

- It starts with a list of URLs to visit, called the **seeds..**
- As the crawler visits these URLs, it identifies all the hyperlinks in the page and adds them to the list of visited URLs, called the **crawl frontier.**
- URLs from the frontier are recursively visited according to a **set of policies.**

Basic crawlers



- This is a **sequential** crawler
- **Seeds** can be any list of starting URLs
- Order of page visits is determined by **frontier** data structure
- **Stop** criterion can be anything

Crawling procedure

- **Simple**
 - Great deal of engineering goes into industry-strength crawlers
 - Industry crawlers crawl a substantial fraction of the Web
 - E.g.: Alta Vista, Northern Lights, HTTtracker
- No guarantee that all accessible Web pages will be located in this fashion
- **Crawler may never halt**
 - pages will be added continually even as it is running.

Crawling overheads

- Delays involved in
 - Resolving the host name in the URL to an IP address using DNS
 - Connecting a socket to the server and sending the request
 - Receiving the requested page in response
- Solution: Overlap the above delays by
 - fetching many pages at the same time

Anatomy of a crawler

- **Page fetching threads**
 - Starts with DNS resolution
 - Finishes when the entire page has been fetched
- **Each page**
 - stored in compressed form to disk/tape
 - scanned for outlinks
- **Work pool of outlinks**
 - maintain network utilization without overloading it
 - Dealt with by *load manager*
- Continue till the crawler has collected a *sufficient* number of pages.

Applications of Web Crawlers

- **Business Intelligence**

Collect information about their competitors and potential collaborators

- **Monitor Web site and Pages of Interest**

A user or community can be notified when new information appears in certain places

- **Support of Search Engine**

Crawlers are the main consumers of Internet bandwidth.

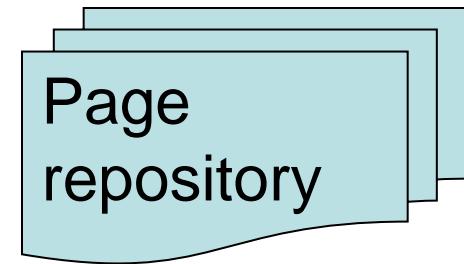
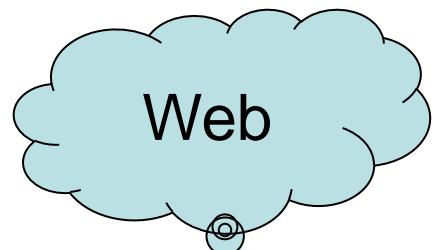
They collect pages for search engines to build their indexes.

Applications of Web Crawlers

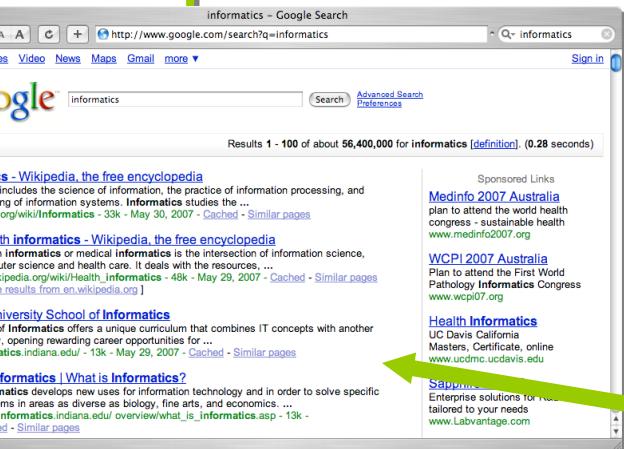
Harvest Email-Address (Malicious Application)

- Used by spammers or collect personal information to be used in **phishing and other identity theft attacks**.
- Well known search engines such as Google, *Yahoo!* and MSN run very efficient **universal crawlers** designed to gather all pages irrespective of their content.
- Other crawlers, sometimes called **preferential crawlers**, are more targeted.
 - They attempt to download only pages of certain **types or topics**.

A crawler within a search engine



Query



Ranker

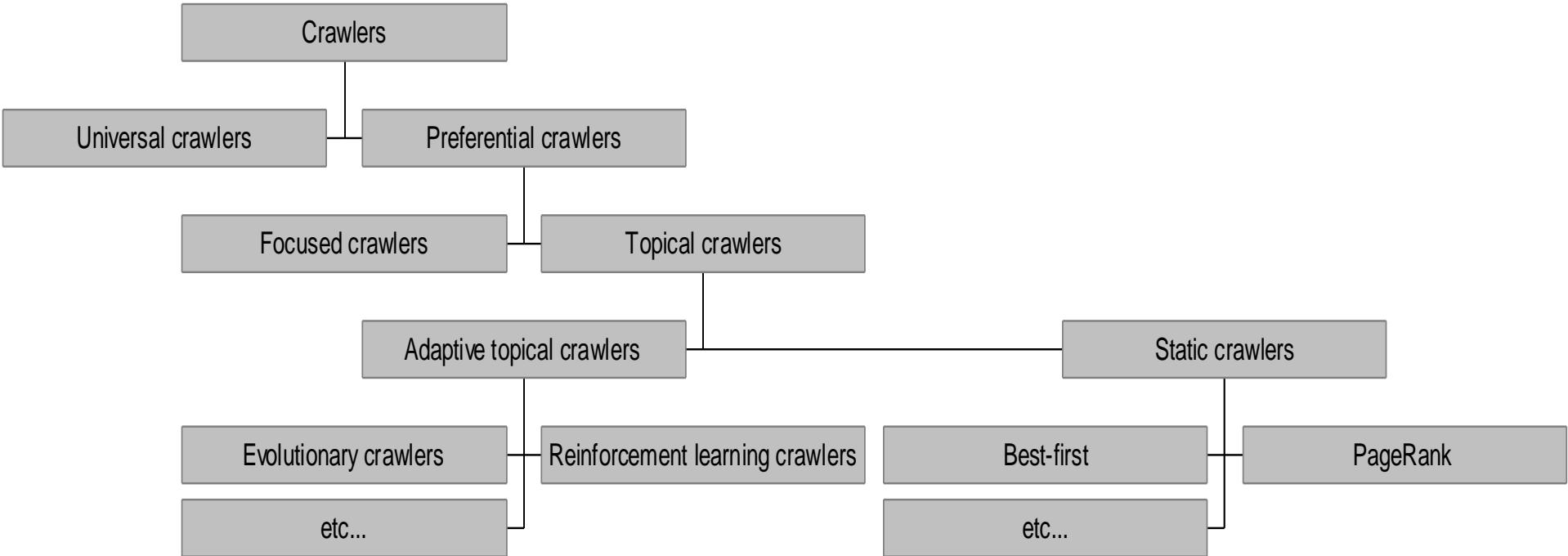
hits

Text & link analysis

Text index

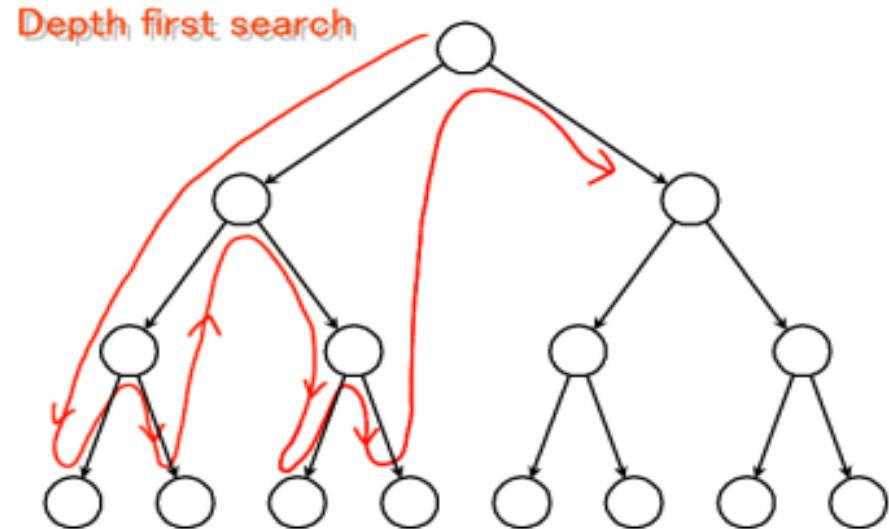
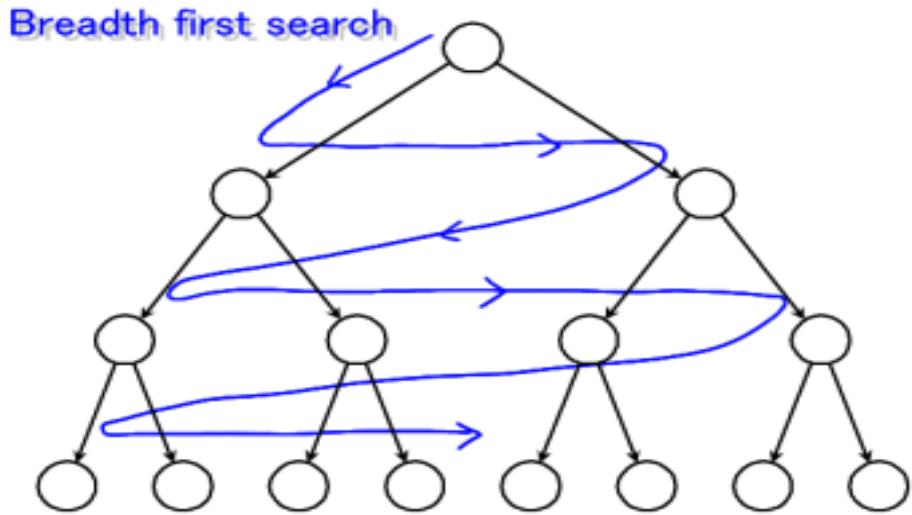
PageRank

One taxonomy of crawlers

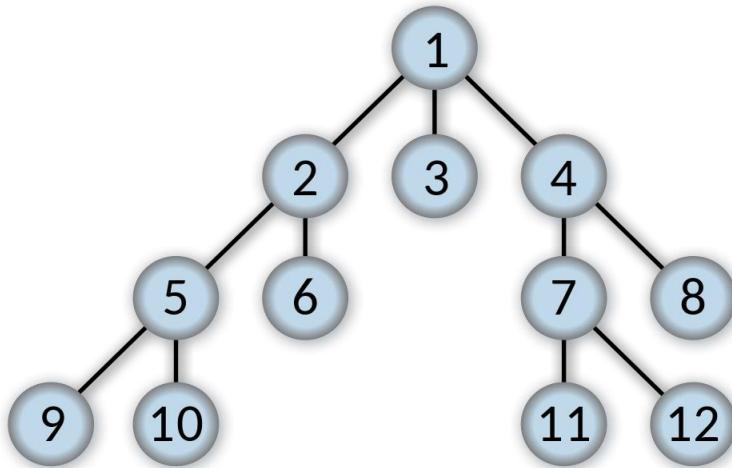


Graph traversal (BFS or DFS?)

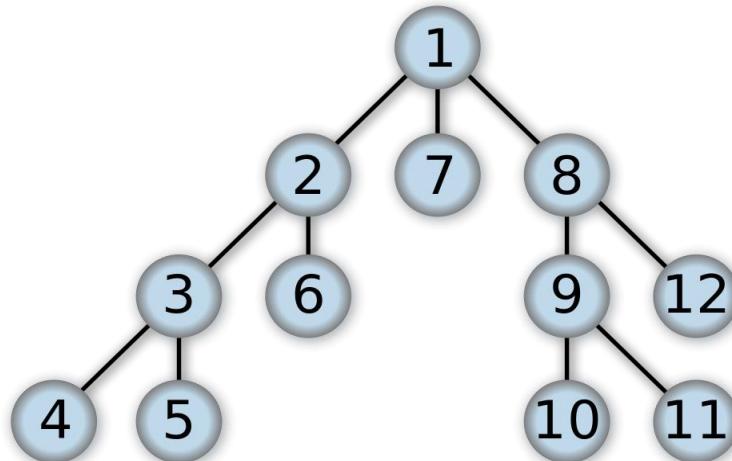
- Breadth First Search
 - Implemented with QUEUE (FIFO)
 - Finds pages along shortest paths
 - If we start with “good” pages, this keeps us close; maybe other good stuff
- Depth First Search
 - Implemented with STACK (LIFO)
 - Wander away (“lost in cyberspace”)



BFS vs DFS



Breadth-first search



Depth-first search

Breadth-First Crawlers

Breadth-First Crawlers

- The frontier may be implemented as a first-in-first-out (FIFO) queue, corresponding to a **breadth-first** crawler.
- The URL to crawl next comes from the **head of the queue** and new URLs are **added** to the **tail of the queue**.
- Once the **frontier reaches its maximum size**, the breadth-first crawler can add to the queue only one unvisited URL from each new page crawled.

Breadth-First Crawlers

- The breadth-first strategy does not imply that pages are visited in “random” order. Because
- The order in which pages are visited by a breadth-first crawler is highly correlated with their **PageRank or indegree values**.
- They are greatly affected by the choice of **seed pages**.
- Topical locality measures indicate that pages in the link neighborhood of a seed page are much more likely to be related to the seed pages than randomly selected pages.

Crawling Algorithm

Initialize queue (Q) with initial set of known URL's.

Until Q empty or page or time limit exhausted:

Pop URL, L, from front of Q.

If L is not an HTML page (.gif, .jpeg, .ps, .pdf, .ppt...)
exit loop.

If already visited L, continue loop(get next url).

Download page, P, for L.

If cannot download P (e.g. 404 error, robot excluded)
exit loop, else.

Index P (e.g. add to inverted index or store cached copy).

Parse P to obtain list of new links N.

Append N to the end of Q.

Implementation issues

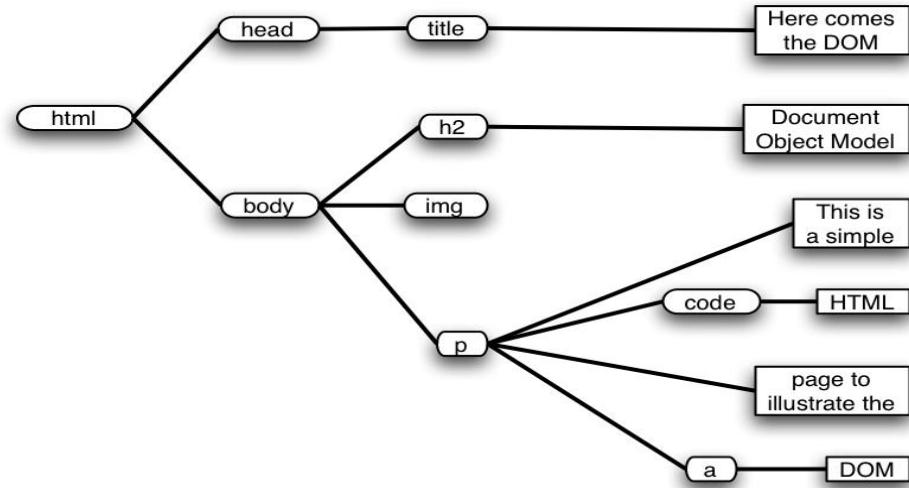
- Don't want to fetch same page twice!
 - Keep lookup table (hash) of visited pages
 - What if not visited but in frontier already?
- The frontier grows very fast!
 - May need to prioritize for large crawls
- Fetcher must be robust!
 - Don't crash if download fails
 - Timeout mechanism
- Determine file type to skip unwanted files
 - Can try using extensions, **but not reliable**
 - Can issue 'HEAD' HTTP commands to get Content-Type (MIME) headers, but overhead of extra Internet requests

More implementation issues

- **Fetching**
 - Get only the first 10-100 KB per page
 - Take care to detect and break redirection loops
 - **Soft fail** for timeout, server not responding, file not found, and other errors

More implementation issues: Parsing

- HTML has the structure of a **DOM (Document Object Model)** tree
- Unfortunately actual HTML is often incorrect in a strict syntactic sense
- Crawlers, like browsers, must be **robust/forgiving**
- Must pay attention to HTML entities and unicode in text
- What to do with a growing number of other formats?
 - Flash, SVG, RSS, AJAX...



More implementation issues

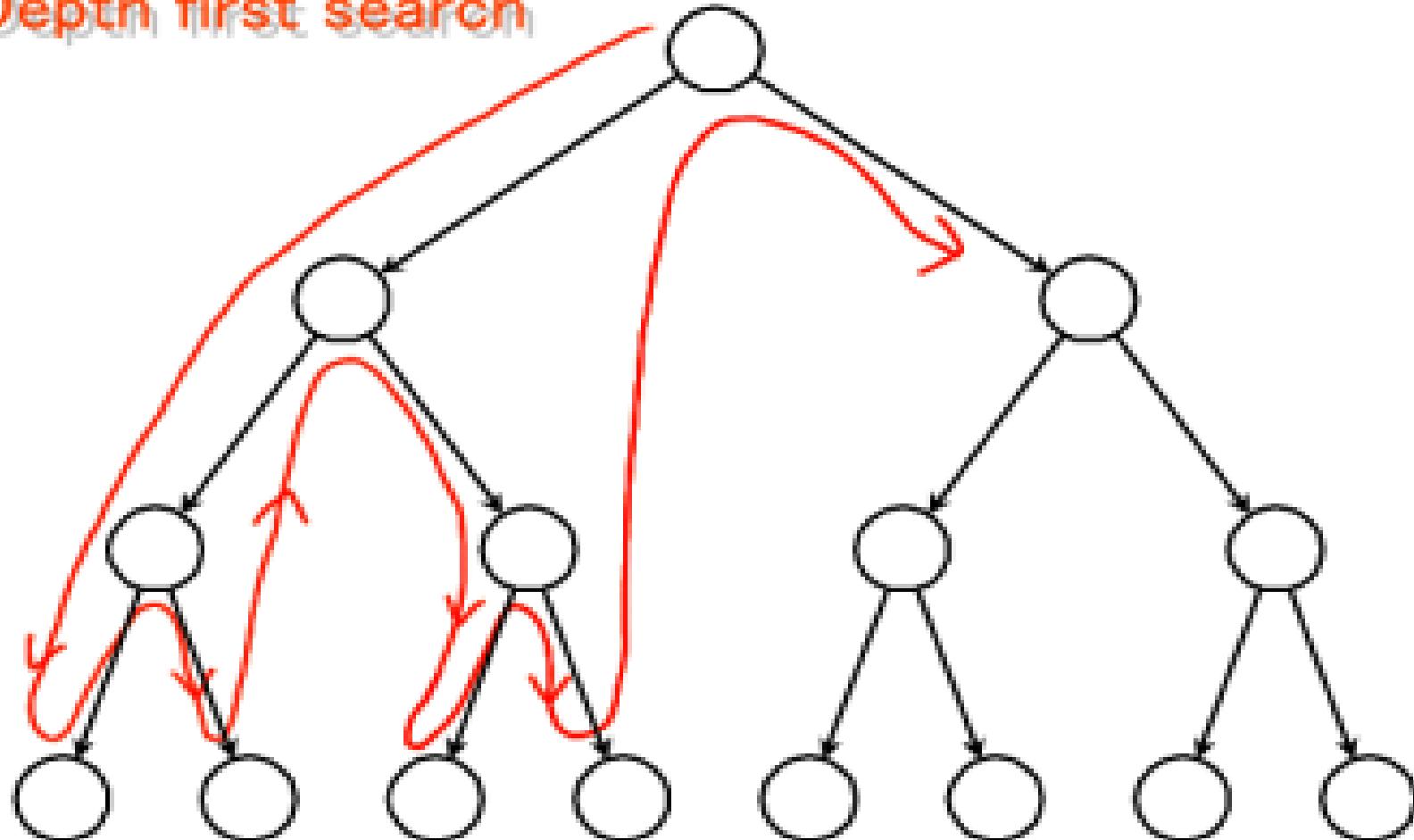
- Stop words
 - Noise words that do not carry meaning should be eliminated (“stopped”) before they are indexed
 - E.g. in English: AND, THE, A, AT, OR, ON, FOR, etc...
 - Typically syntactic markers
 - Typically the most common terms
 - Typically kept in a negative dictionary
 - 10–1,000 elements
 - E.g. http://ir.dcs.gla.ac.uk/resources/linguistic_utils/stop_words
 - **Parser** can detect these right away and disregard them

Depth-First Crawlers

- The frontier may be implemented as a last-in-first-out (LIFO) stack, corresponding to a **depth-first** crawler.
- Wander away (“lost in cyberspace”) Use depth first search (DFS) algorithm
- Get the 1st link not visited from the start page
- Visit link and get 1st non-visited link
- Repeat above step till no non-visited links
- Go to next non-visited link in the previous level and repeat 2nd step

Depth-First Crawlers

Depth first search



Breadth-First vs. Depth-First Crawlers

- breadth-first is more careful by checking all alternatives
- complete and optimal
- very memory-intensive
- depth-first goes off into one branch until it reaches a leaf node
- not good if the goal node is on another branch
- neither complete nor optimal
- uses much less space than breadth-first

WEB MINING

Preferential Crawlers

Preferential Crawlers

- A different crawling strategy is obtained if the frontier is implemented as a **priority queue** rather than a FIFO queue.
- **Preferential crawlers** assign each unvisited link a priority based on an estimate of the value of the **linked page**.
- **The estimate can be based on**
 - Topological properties
(e.g., the indegree of the target page)
 - Content properties (e.g., the similarity between a user query and the source page)
 - Combination of measurable features.

Preferential Crawlers

- If pages are visited in the order specified by the **priority values** in the frontier, then we have a **best-first** crawler.
- Best First Search is a **heuristic** based search algorithm.
- In this approach, **relevancy calculation** is done for **each link** and the most relevant link, such as one with the highest relevancy value, is fetched from the frontier.
- Thus every time the best available link is **opened and traversed**.

Preferential Crawlers

- The **time complexity** of inserting a URL into the priority queue is $O(\log F)$, where F is the frontier size (looking up the hash requires constant time).
- To dequeue a URL, it must first be removed from the priority queue ($O(\log F)$) and then from the **hash table** (again $O(1)$).
- Thus the parallel use of the two data structures yields a logarithmic total **cost per URL**.
- Once the frontier's **maximum size** is reached, only the best URLs are kept; the frontier must be pruned after each new set of links is added.

Preferential crawling algorithms: Examples

- **Breadth-First**
 - Exhaustively visit all links in order encountered
- **Best-N-First**
 - Priority queue sorted by similarity, explore top N at a time
 - Variants: DOM context
- **PageRank**
 - Priority queue sorted by keywords, PageRank
- **SharkSearch**
 - Priority queue sorted by combination of similarity, anchor text, similarity of parent, etc. (powerful cousin of FishSearch)
- **InfoSpiders**
 - Adaptive distributed algorithm using an evolving population of learning agents

Fish Search

- Fish Search is a dynamic heuristic search algorithm.
- It works on the intuition that **relevant links** have **relevant neighbours**;
- hence it **starts** with a **relevant link** and goes deep under that link and stops searching under the links that are irrelevant.
- The key point of Fish Search algorithm lies in the maintenance of **URL order**.

A* Search

- A* uses Best First Search.
- It calculates the relevancy of each link and the difference between expected relevancy of the target web-page and the current link.
- The total of these two values serve as the measure for selecting the best path.

Adaptive A* Search

- Adaptive A* Search works on **informed heuristics** to focus its searches.
- With each **iteration**, it updates the **relevancy value** of the page and uses it for the **next traversal**.
- The pages are updated for **$\log(\text{Graph Size})$** times, (**$\log(\text{Graph Size})$** times the overhead of updating is much more than the improvement that can be achieved in getting more relevant pages).

More implementation issues

Conflation and thesauri

- Idea: improve **recall** by merging words with **same meaning**
 1. We want to ignore superficial **morphological** features, thus merge semantically similar tokens
 - {student, study, studying, studious} => studi
 2. We can also conflate **synonyms** into a single form using a thesaurus
 - 30-50% smaller index
 - Doing this in both pages and queries allows to retrieve pages about ‘automobile’ when user asks for ‘car’
 - Thesaurus can be implemented as a hash table

More implementation issues

- **Stemming**
 - In linguistic morphology and information retrieval, stemming is the process of **reducing inflected** (or sometimes derived) words to **their word stem, base or root form.**
 - A stemming algorithm might also reduce the words *fish**ing*, *fishe**d*, and *fisher* to the stem ***fish***.

More implementation issues

- **Static vs. dynamic pages**

- Is it worth trying to eliminate dynamic pages and only index static pages?
- Examples:
 - <http://www.census.gov/cgi-bin/gazetteer>
 - <http://informatics.indiana.edu/research/colloquia.asp>
 - <http://www.amazon.com/exec/obidos/subst/home/home.html/02-8332429-6490452>
 - <http://www.imdb.com/Name?Menczer,+Erico>
 - <http://www.imdb.com/name/nm0578801/>
- Why or why not? How can we tell if a page is dynamic?
What about '**spider traps**'?
- What do Google and other search engines do?

More implementation issues

- **Relative vs. Absolute URLs**

- Crawler must translate relative URLs into absolute URLs
- Need to obtain Base URL from HTTP header, or HTML Meta tag, or else current page path by default
- Examples
 - **Base:** `http://ww.vit.ac.in/schools.html`
 - **Relative URL:** `schools.html`
 - **Absolute URL:** `https://www.vit.ac.in/schools.html`

More implementation issues

- **URL canonicalization**
 - All of these:
 - <http://www.cnn.com/TECH>
 - <http://WWW.CNN.COM/TECH/>
 - <http://www.cnn.com:80/TECH/>
 - <http://www.cnn.com/bogus/.../TECH/>
 - Are really equivalent to this canonical form:
 - <http://www.cnn.com/TECH/>
 - In order to avoid duplication, the crawler must transform all URLs into canonical form
 - Definition of “canonical” is arbitrary, e.g.:
 - Could always include port
 - only include port when not default :80

More implementation issues

- Spider traps
 - **Misleading sites:** indefinite number of pages dynamically generated by **CGI scripts**
 - Paths of arbitrary depth created using soft directory links and path rewriting features in HTTP server
 - Only heuristic defensive measures:
 - Check URL length; assume spider trap above some threshold, for example 128 characters
 - Watch for sites with very large number of URLs
 - Eliminate URLs with non-textual data types
 - May disable crawling of dynamic pages, if can detect

More implementation issues

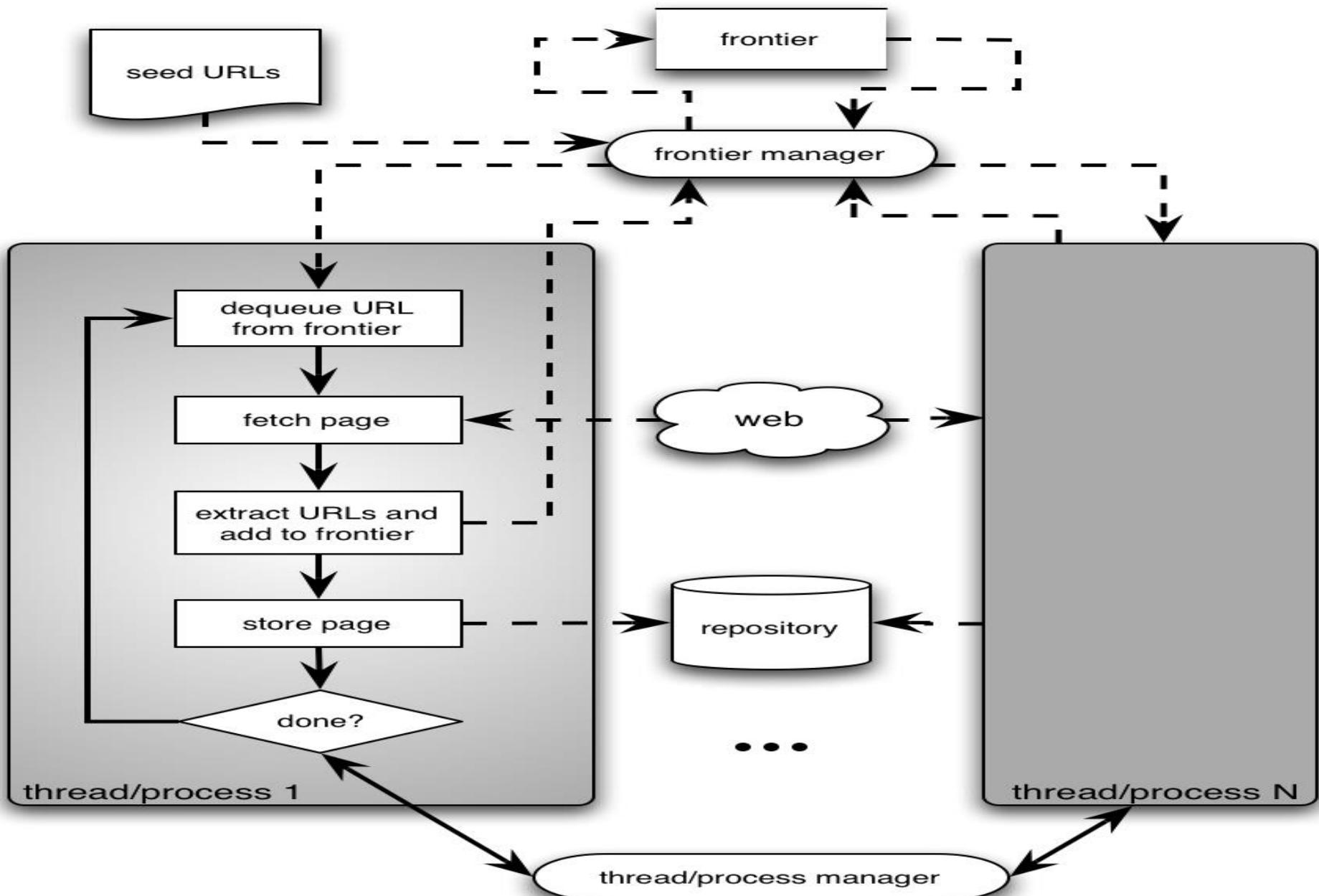
- **Page repository**

- Naïve: store each page as a separate file
 - Can map URL to unique filename using a hashing function
 - This generates a huge number of files, which is inefficient from the storage perspective
- Better: combine many pages into a single large file, using some XML markup to separate and identify them
 - Must map URL to {filename, page_id}
- Database options
 - Any RDBMS -- large overhead
 - Light-weight, embedded databases

Concurrency

- A crawler incurs several delays:
 - Resolving the host name in the URL to an IP address using DNS
 - Connecting a socket to the server and sending the request
 - Receiving the requested page in response
- Solution: Overlap the above delays by **fetching many pages concurrently**

Architecture of a concurrent crawler



Concurrent crawlers

- Can use **multi-processing** or **multi-threading**
- Each process or thread works like a sequential crawler, except they **share data structures**: frontier and repository
- Shared data structures must be synchronized (locked for concurrent writes)
- Speedup of factor of 5-10 are easy this way

WEB MINING

Universal Crawlers – Focused Crawlers

What is Universal Crawler ?

- Universal crawler is a **fast** precise and **reliable** Internet crawler.
- Methodical, automated manner and creates the index of documents

How UC works?

- Uc crawler Downloads first website.
- It goes through html, finds an **link tag**, and retrieves **outside link**.
- When it finds "`<ahref="http://www.sourceforge.net">` website ``" it copies the link and adds it to a **list of pages it plans to crawl**.
- It crawls it and analyzes it again
- Each website is saved into "web" folder, so there is **no need** to **re-download pages** if job was stopped.

Universal crawlers : Basics

- Support universal search engines
- Large-scale
- Huge cost (network bandwidth) of crawl is amortized over many queries from users
- **Incremental updates** to existing index and other data repositories

Large-scale universal crawlers

Two major issues:

1. Performance

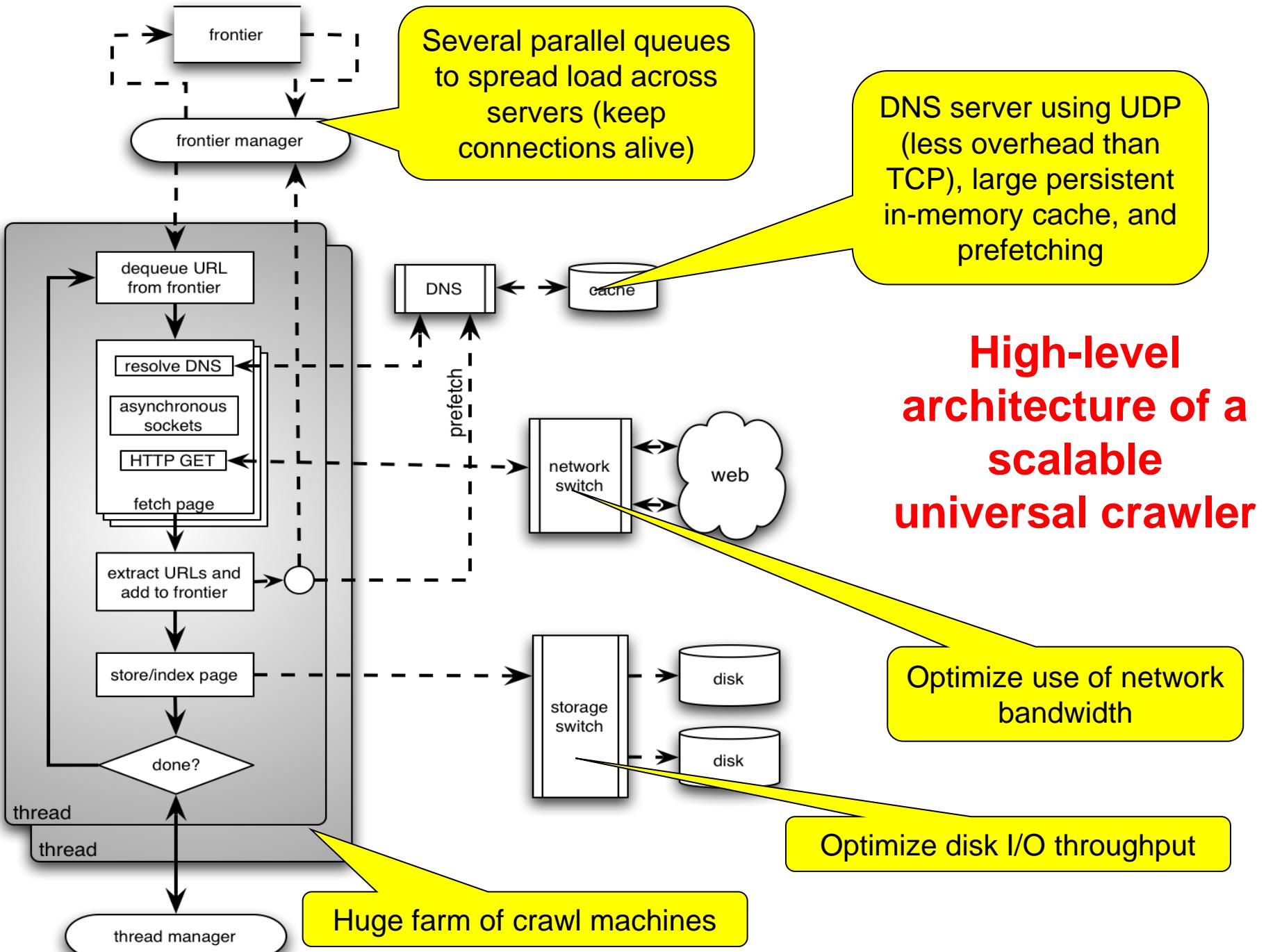
- Need to scale up to billions of pages

2. Policy

- Need to trade-off coverage, freshness, and bias (e.g. toward “important” pages)

Large-scale crawlers: scalability

- Need to minimize overhead of **DNS lookups**
- Need to optimize utilization of network bandwidth and disk throughput (I/O is bottleneck)
- Use **asynchronous sockets**
 - **Multi-processing or multi-threading** do not scale up to billions of pages
 - **Non-blocking:** hundreds of network connections open simultaneously
 - **Polling socket** to monitor completion of network transfers



Universal crawlers: Policy

- Coverage
 - New pages get added all the time
 - Can the crawler find every page? Yes
- Freshness
 - Pages change over time, get removed, etc.
 - How frequently can a crawler revisit ? Timeout
- Trade-off!
 - Focus on most “important” pages (crawler bias)?
 - “Importance” is subjective

Estimating page change rates

- Algorithms for maintaining a crawl in which most pages are fresher than a specified.
- Assumption: predicts the future
 - Frequency of change not a good predictor
 - Degree of change is a better predictor

Do we need to crawl the entire Web?

- If we cover too much, it will get **stale**
- There is an abundance of pages in the Web
- For PageRank, pages with very low prestige are **largely useless**
- **What is the goal?**
 - *General search engines: pages with **high prestige***
 - *News portals: pages that **change often***
 - *Vertical portals: pages on some topic*

Benefits of UC

- Get local copy of crawled pages.
- Get link analysis on crawled pages using Page Rank.
- Resume download only on pages not crawled

FOCUSED CRAWLERS

Focussed Crawlers

- Crawl only pages in **certain categories**
- A focused Crawler attempts to bias the crawler towards pages in some categories in which user is interested.
- One application of a such preferential crawler would be to maintain a **web taxonomy** such as Yahoo! Directory (dir.yahoo.com) or the volunteer based Open Directory Project (ODP, demoz.org).

Functionality of the Classifier

- The **classifier** would guide the crawler by preferentially selecting from **frontier pages** that appear most likely to belong to categories of interest, according to **classifier's prediction**.

Focused crawler Components

The focused crawler has three main components:

- 1. Classifier**
- 2. Distiller**
- 3. Crawler**

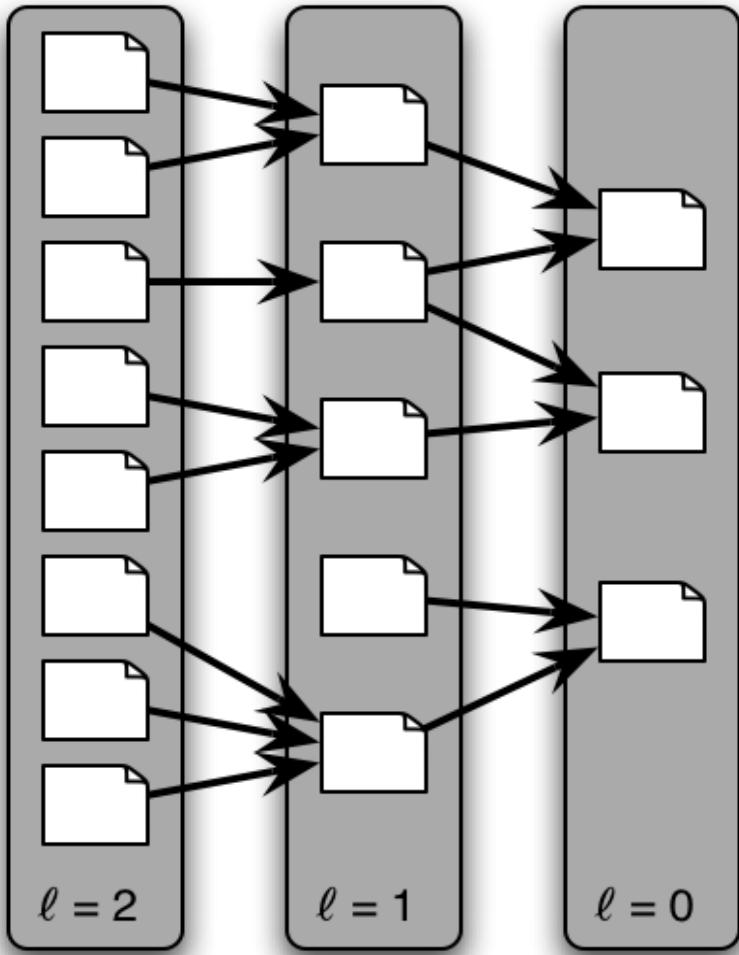
Focused crawlers

- Can have **multiple topics** with as many classifiers, with scores appropriately combined
- Can use a **distiller** to find topical hubs periodically, and add these to the frontier
- Can use alternative classifier algorithms to naïve-Bayes, e.g. **SVM** and **neuralnets** have reportedly performed better

Soft Focused Strategy & Hard Focused Strategy

- **Soft Focused Strategy** : the crawler uses the score $R(p)$ of each crawled page p as a **priority value** of each crawled page p as a priority value for all unvisited URLs extracted from p . The URLs are added to the frontier.
- **Hard Focused Strategy** : For a crawled page p , the classifier first finds the **leaf category** $C(p)$ in the taxonomy most likely to include p . If an ancestor of $C(p)$ is a focus category ,then URLs from the crawled page p are added to the frontier otherwise they are discarded.

Context-focused crawlers



Context graph

- Same idea, but **multiple classes** (and classifiers) based on **link distance** from relevant targets
 - $\ell=0$ is topic of interest
 - $\ell=1$ link to topic of interest
 - Etc.
- Initially needs a back-crawl from seeds (or known targets) to train classifiers to estimate distance
- Links in frontier prioritized based on estimated distance from targets
- Outperforms standard focused crawler empirically

TOPICAL CRAWLERS

Topical crawlers : Introduction

- A way to address the scalability limitations of universal search engines, by distributing the crawling process across **users, queries or even client computers.**
- Can guide the **navigation of links** with the goal of efficiently locating highly **relevant target pages**

Topical crawlers

- It is a topic (query, description, keywords) and a set of seed pages (not necessarily relevant)
- No labeled examples
- Must predict relevance of **unvisited links** to **prioritize**
- Original idea: Menczer 1997, Menczer & Belew 1998

Topical Crawler

- Example:-**MySpiders applet** (myspiders.informatics.indiana.edu). The applet is designed to demonstrate two topical crawling algorithms, **best-N-first** and **InfoSpiders**
- Unlike a search engine, this application has **no index to search for results**.
- Instead the Web is crawled in real time. As pages deemed relevant are crawled, they are displayed in a list that is kept sorted by a user-selected criterion: **score or recency**.
- The **score** is simply the content (cosine) similarity between a page and the query, and the **recency** of a page is estimated by the last-modified header, if returned by the server (not a very reliable estimate).

Pros and Cons of Topical Crawler

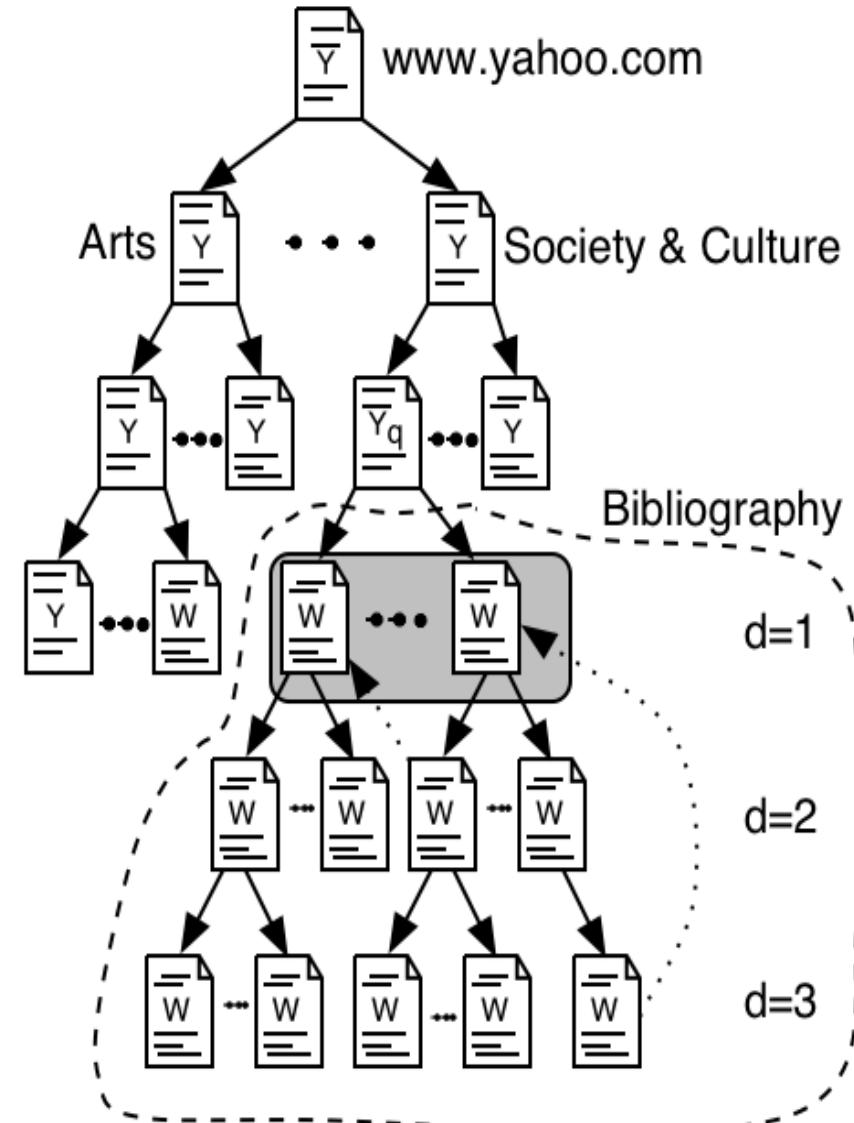
- All **hits** are fresh by definition.
- This makes this type of crawlers suitable for applications that look for very **recently posted documents**, which a search engine may not have indexed yet.
- On the **down side**, the search is **slow compared** to a traditional search engine because the user has to wait while the **crawler fetches and analyzes pages**.
- Another disadvantage is that the **ranking algorithms** cannot take advantage of global **prestige measures**, such as **PageRank**, available to a traditional search engine.

Topical locality

- Topical locality is a **necessary** condition for a topical crawler to work, and for surfing to be a worthwhile activity for humans
- Links must encode **semantic** information, i.e. say something about neighbor pages, not be random
- It is also a **sufficient** condition if we start from “good” seed pages
- Crawling algorithms can use words and hyperlinks, associated respectively with a **lexical and a link topology**.
- In the former, **two pages** are close to each other if they have **similar textual content**; in the latter, if there is a **short path between them**

Quantifying topical locality

- Different ways to pose the question:
 - How quickly does semantic **locality** decay?
 - How fast is topic drift?
 - How quickly does **content change** as we surf away from a starting page?
- To answer these questions, let us consider **exhaustive** crawls
TECHNIQUES



Topical locality

- Link-content conjecture
- link-cluster conjecture:

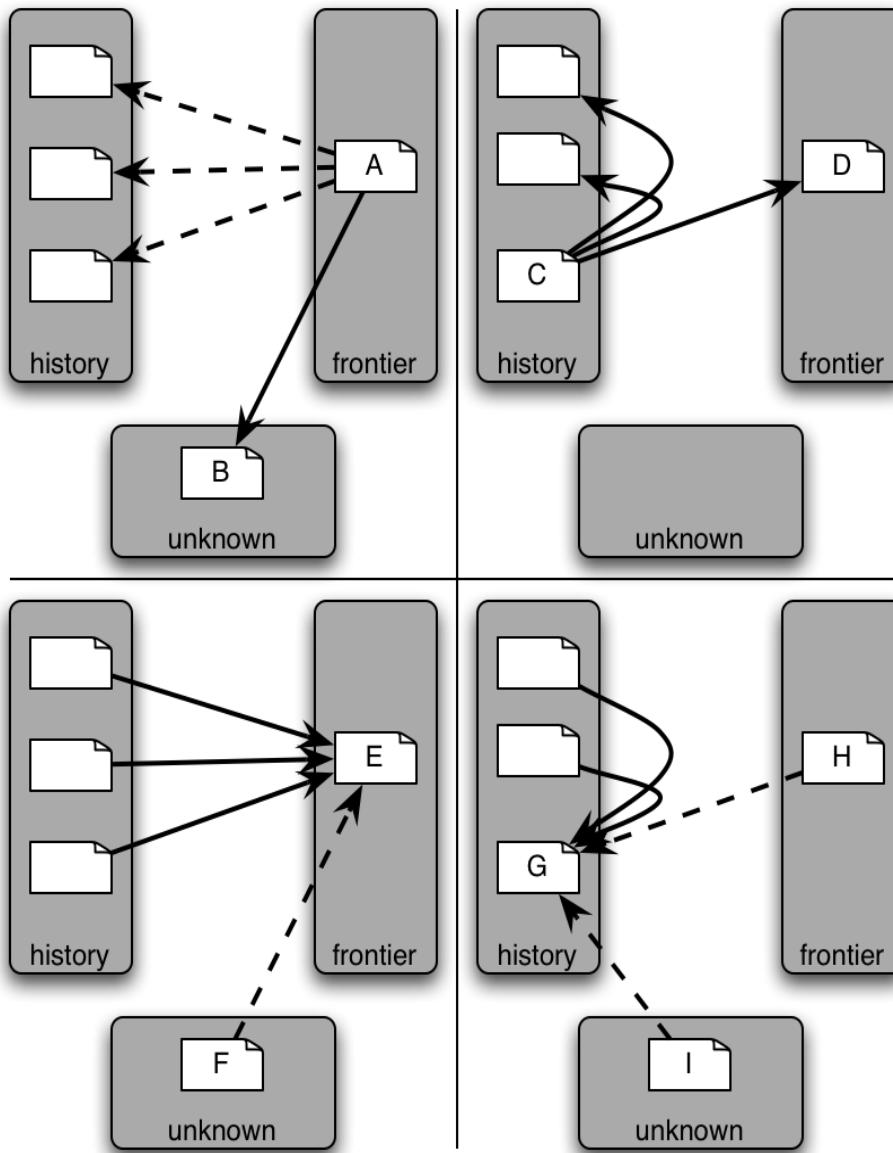
Link-content conjecture

- whether two pages that link to each other are more likely to be **lexically** similar to each other, compared to two **randomly selected pages**.
- This makes the assumption that pages which link to each other are closely **topically related**,
- *i.e. a page on bananas is likely to link to other pages about bananas (or at least fruit).*

Link-cluster conjecture

- **link-cluster conjecture:** whether two pages that link to each other are more likely to be semantically related to each other, compared to two randomly selected pages.
- This assumes that pages which are clustered together, or in the same "**web-community**" are closely topically related, *i.e. the page about bananas from above links to a page about fruit which links to a page about food - all three are closely topically related.*

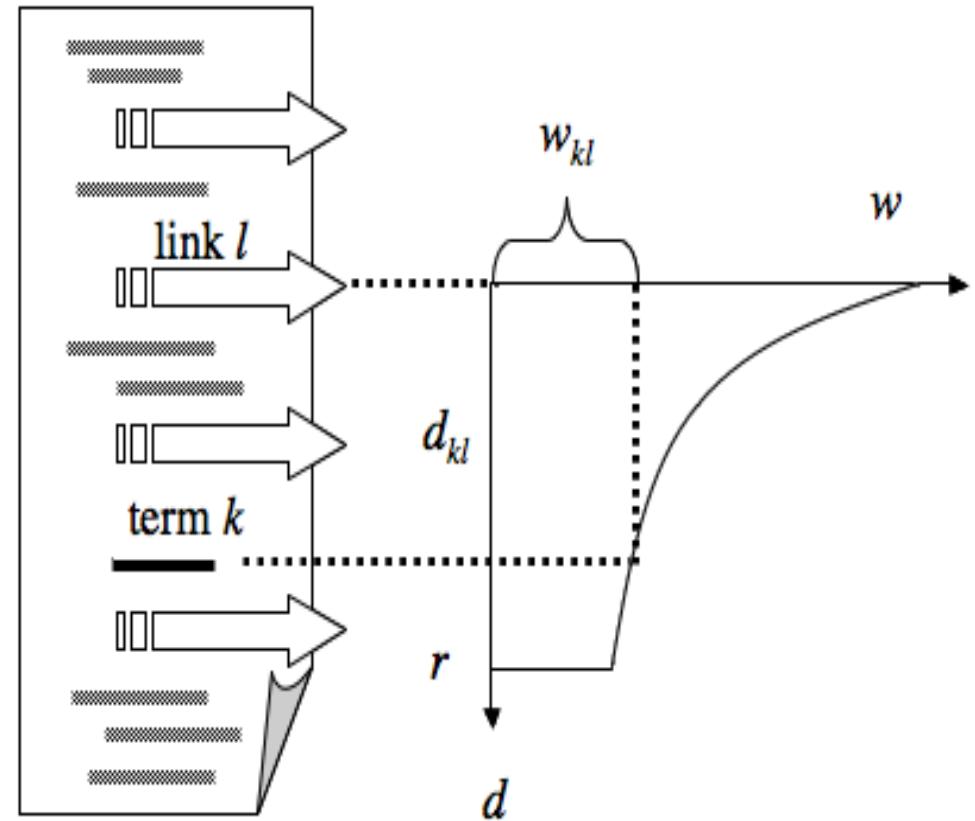
Topical locality-inspired for topical crawlers



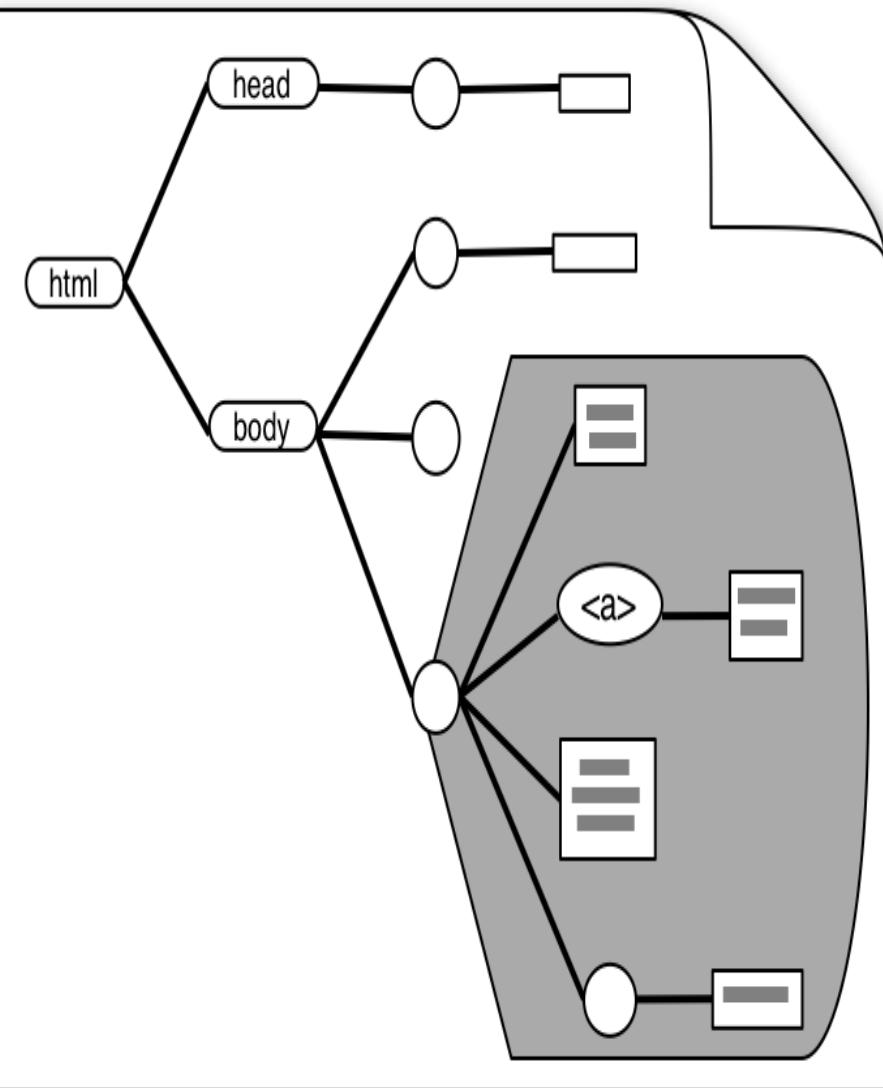
- Co-citation (a.k.a. **sibling locality**): A and C are good **hubs**, thus A and D should be given **high priority**
- Co-reference (a.k.a. **bibliographic coupling**): E and G are good **authorities**, thus E and H should be given **high priority**

Link context based on text neighborhood

- Often consider a fixed-size window, e.g. 50 words around anchor
- Can weigh links based on their distance from topic keywords within the document ?
- **YES, (*InfoSpiders, Clever*)**
- **Anchor text deserves extra importance**



Link context based on DOM tree



- Consider DOM subtree rooted at parent node of link's `<a>` tag
- Or can go further up in the tree (Naïve Best-First is special case of entire document body)

About Exelixis

Exelixis, Inc. is a leading genomics-based drug discovery company focused on product development through its expertise in comparative genomics and model system genetics. These technologies provide a rapid, efficient and cost effective way to move from DNA sequence data to knowledge about the function of genes and the proteins they encode. The company's technology is broadly applicable to all life sciences industries including pharmaceutical, diagnostic, agricultural biotechnology and animal health. Exelixis has partnerships with Aventis CropScience S.A., Bayer Corporation, Bristol-Myers Squibb Company, Elan Pharmaceuticals, Inc., Pharmacia Corporation, Protein Design Labs, Inc., Scios Inc. and Dow AgroSciences LLC, and is building its internal development program in the area of oncology. For more information, please visit the company's web site at www.exelixis.com.

<P class=MsoNormal>
About Exelixis
Exelixis, Inc. is a leading genomics-based drug discovery company focused on product development through its expertise in comparative genomics and model system genetics. These technologies provide a rapid, efficient and cost effective way to move from DNA sequence data to knowledge about the function of genes and the proteins they encode. The company's technology is broadly applicable to all life sciences industries including pharmaceutical, diagnostic, agricultural biotechnology and animal health. Exelixis has partnerships with Aventis CropScience S.A., Bayer Corporation, Bristol-Myers Squibb Company, Elan Pharmaceuticals, Inc., Pharmacia Corporation, Protein Design Labs, Inc., Scios Inc. and Dow AgroSciences LLC, and is building its internal development program in the area of oncology. For more information, please visit the company's web site at
www.exelixis.com.<o:p></o:p>
</P>

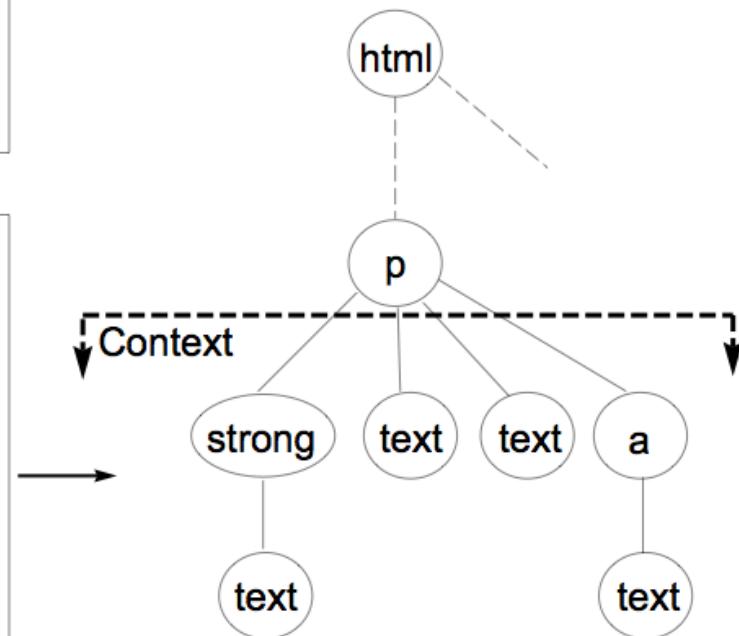
- <p>
-
 <text>about exelixis</text>

<text>exelixis inc is a leading genomics based drug discovery company focused on product development through its expertise in comparative genomics and model system genetics these technologies provide a rapid efficient and cost effective way to move from dna sequence data to knowledge about the function of genes andthe proteins they encode the company s technology is broadly applicable to all life sciences industries including pharmaceutical diagnostic agricultural biotechnology and animal health exelixis has partnerships with aventis cropscience s a bayer corporation bristol myers squibb company elan pharmaceuticals inc pharmacia corporation protein design labs inc scios inc and dow agrosciences llc and is building its internal development program in the area of oncology</text>
<text>for more information please visit the company s web site at</text>
-
 <text>www exelixis com</text>

</p>

DOM context

Link score = linear combination between page-based and context-based similarity score

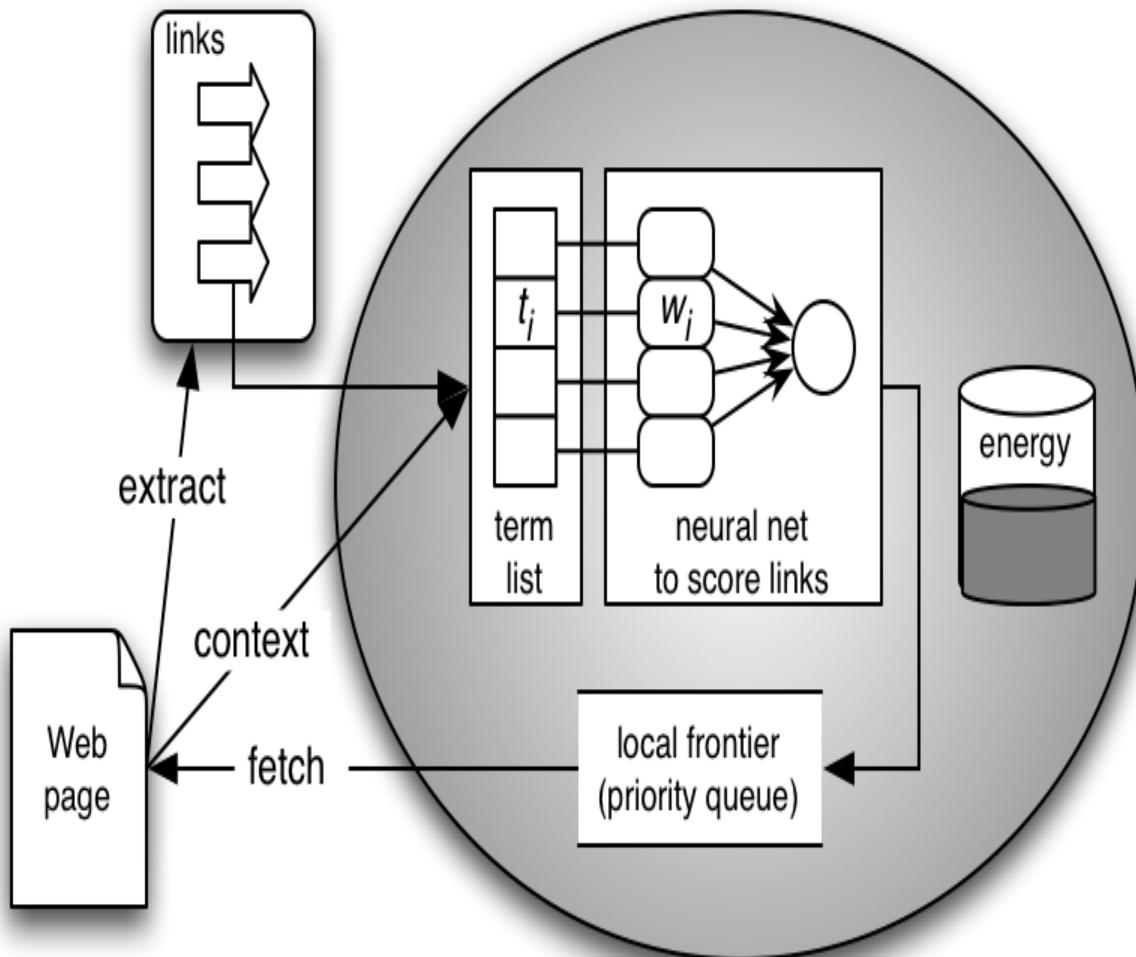


Exploration vs Exploitation

- **Best- N -First** (or BFS N)
- Rather than re-sorting the frontier every time you add links, be lazy and sort only every N pages visited
- Empirically, *being less greedy* helps crawler performance significantly: escape “local topical traps” by exploring more

InfoSpiders

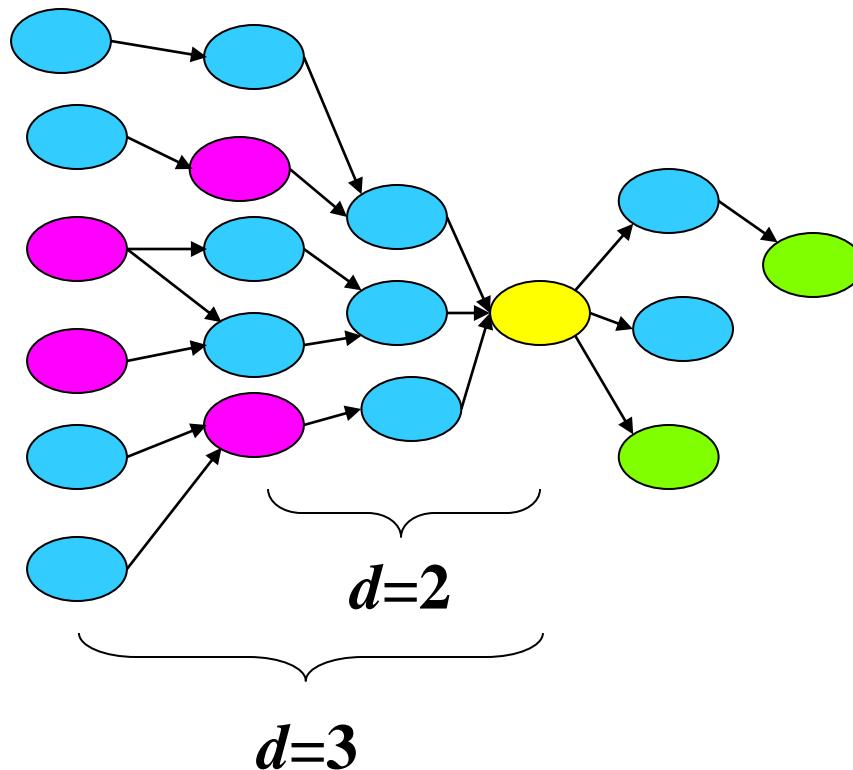
- A series of intelligent multi-agent topical crawling algorithms employing various adaptive techniques:
 - Evolutionary bias of exploration/exploitation
 - Selective query expansion
 - (Connectionist) reinforcement learning



Evaluation of topical crawlers

- Goal: build “better” crawlers to support applications
- Build an unbiased evaluation framework
 - Define common tasks of measurable difficulty
 - Identify topics, relevant targets
 - Identify appropriate performance measures
 - Effectiveness: quality of crawler pages, order, etc.
 - Efficiency: separate CPU & memory of crawler algorithms from bandwidth & common utilities

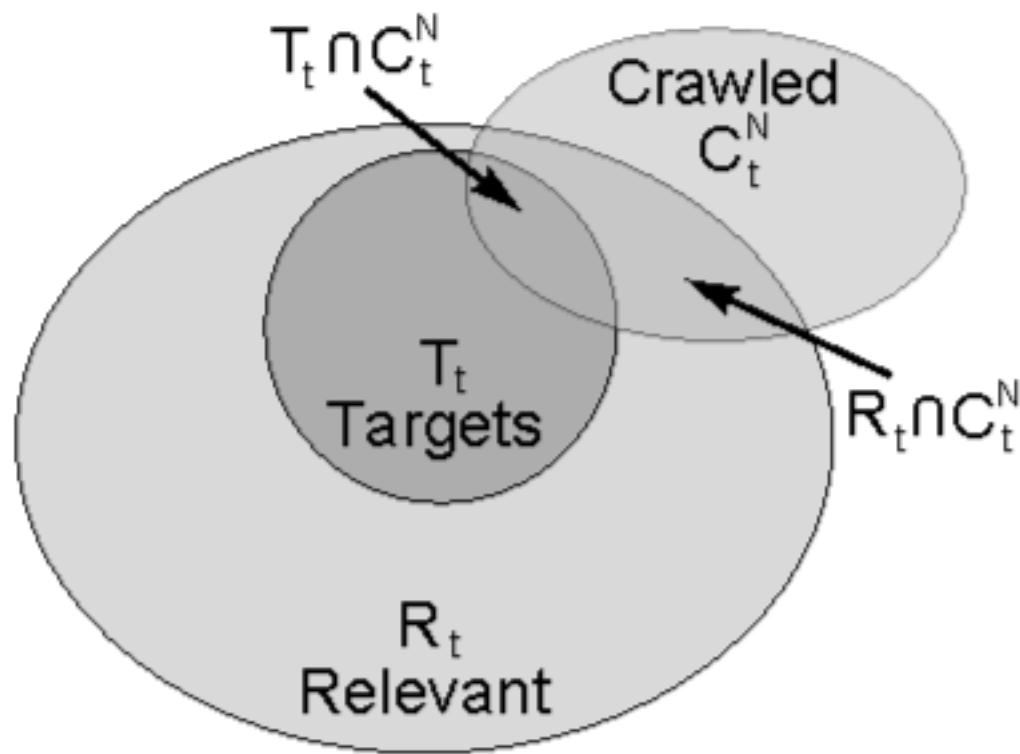
Tasks



Start from
seeds, find
targets
and/or pages
similar to
target
descriptions

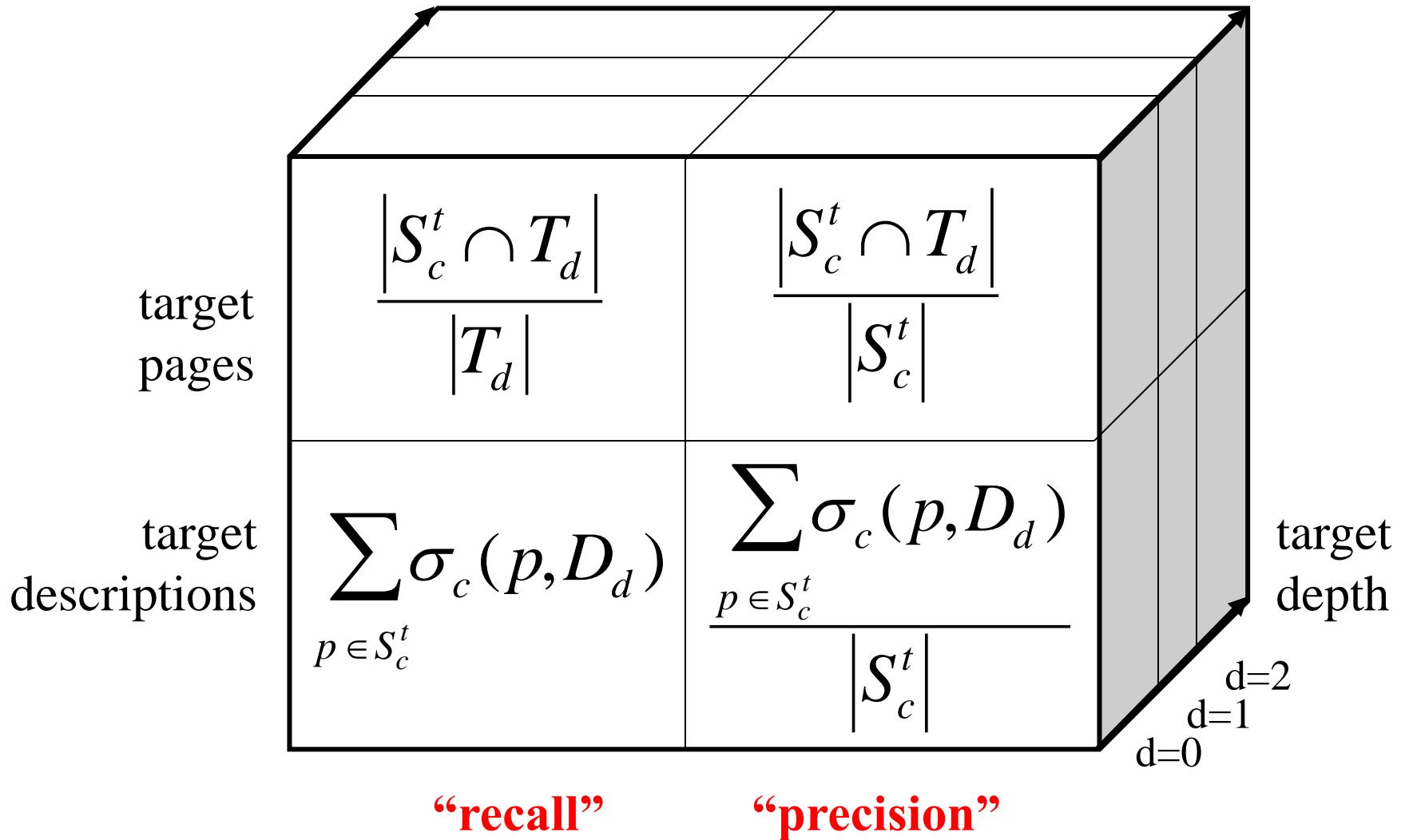
Back-crawl from targets to get seeds

Target based performance measures



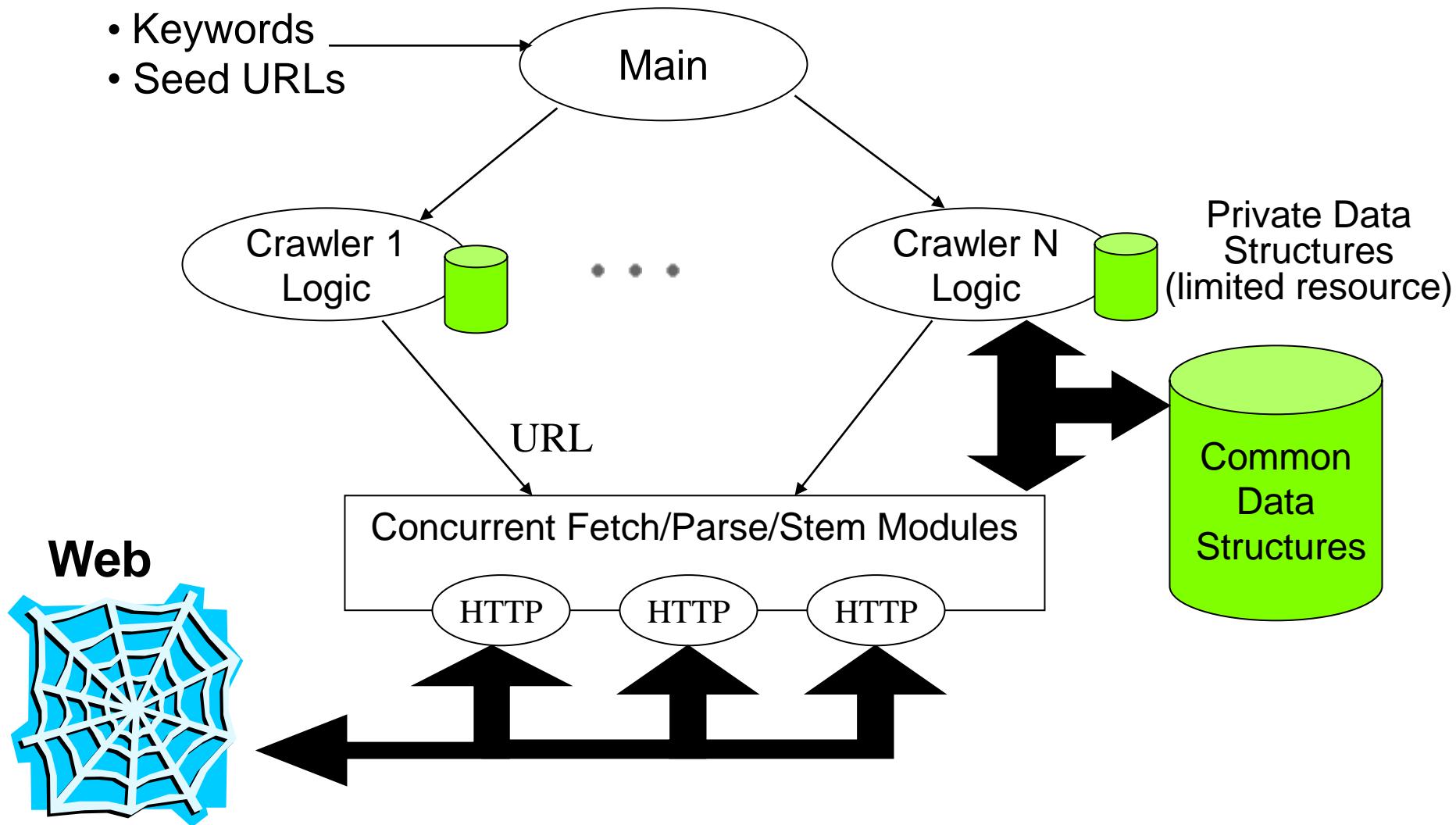
Q: What assumption are we making? A: Independence!...

Performance matrix



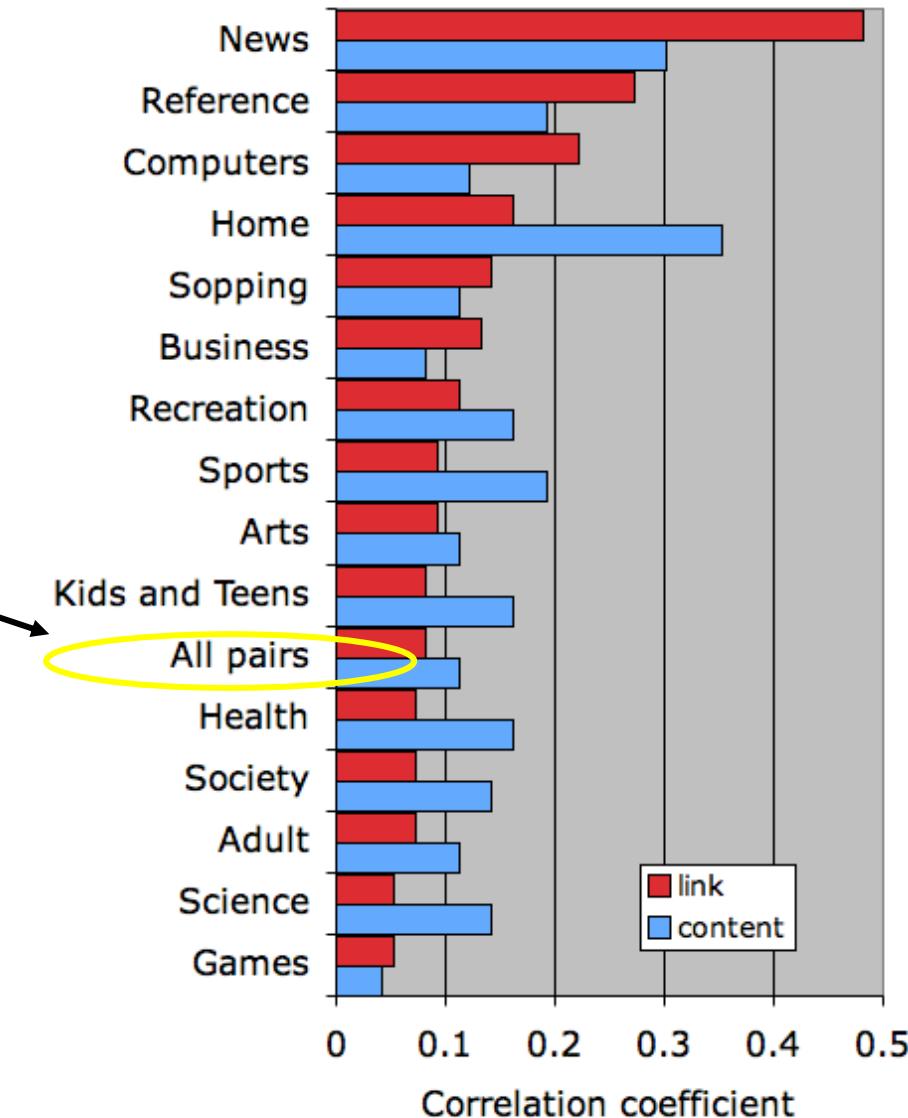
Crawling evaluation framework

- Keywords
- Seed URLs



Correlations between different similarity measures

- Semantic similarity measured from ODP, correlated with:
 - Content similarity: **TF or TF-IDF vector cosine**
 - Link similarity: **Jaccard coefficient** of (in+out) link neighborhoods
- Correlation overall is **significant but weak**
- Much stronger topical locality NEEDED



WEB MINING

Crawlers Evaluation –Inverted Indexing

TF - IDF

- **TF-IDF**, short for **Term Frequency–Inverse Document Frequency**, is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus.
- Variations of the **TF–IDF weighting scheme** are often used by search engines as ***a central tool in scoring and ranking a document's relevance*** given as user query.
- TF-IDF is one of the most popular term-weighting schemes.
- For instance, 83% of text-based recommender systems in the domain of digital libraries use **TF-IDF**.

TF - IDF

- Suppose we have a set of English text documents and wish to determine which document is most relevant to the query "**the brown cow**".
- A simple way to start out is by **eliminating documents** that do not contain all three words "the", "brown", and "cow", but this still leaves many documents.
- To further distinguish them, we might count the number of times each term occurs in each document; *the number of times a term occurs in a document is called its term frequency.*

TF - IDF

- Because the term "the" is so common, term frequency will tend to incorrectly emphasize documents which happen to use the word "**the**" more frequently, without giving enough weight to the more meaningful terms "**brown**" and "**cow**".
- The term "the" is not a good keyword to distinguish **relevant and non-relevant documents** and terms, unlike the less common words "brown" and "cow".
- Hence an ***inverse document frequency*** factor is incorporated which diminishes the weight of terms that occur very frequently in the document set and increases the weight of terms that occur rarely.

The TFIDF Encoding

(Term Frequency x Inverse Document Frequency)

A *term* is a word, or some other frequently occurring item

Given some term i , and a document j , the *term count* is the number of times that term i occurs in document j

Given a collection of k terms and a set D of documents, the *term n_{ij} frequency*, is:

$$tf_{ij} = \frac{n_{ij}}{\sum_{k=1}^T n_{kj}}$$

considering only the terms of interest, this is the proportion of document j that is made up from term i .

TF - IDF

Term frequency tf_{ij} is a measure of the importance of term i in document j

Inverse document frequency (which we see next) is a measure of the *general* importance of the term.

I.e. **High term frequency** for “apple” means that apple is an important word in a specific document.

But **high document frequency** (low inverse document frequency) for “apple”, given a particular set of documents, means that apple is not all that important overall, since it is in all of the documents.

TF - IDF

Inverse document frequency of term i is:

$$idf_i = \log \frac{|D|}{\{d_j : d_j \in D\}}$$

Log of: ... the number of documents in the master collection,
divided by the number of those documents that contain the term.

TFIDF encoding of a document

So, given:

- a background collection of documents
 - (e.g. 100,000 random web pages,
 - all the articles we can find about cancer
 - 100 student essays submitted as coursework
- a specific ordered list (possibly large) of terms

We can encode any document as a vector of TFIDF numbers, where the i th entry in the vector for document j is:

$$tf_{ij} \times idf_i$$

Turning a document into a vector

Suppose our Master List is:

(banana, cat, dog, fish, read)

Suppose document 1 contains only:

“Bananas are grown in hot countries, and cats like bananas.”

And suppose the *background frequencies* of these words in a large random collection of documents is $(0.2, 0.1, 0.05, 0.05, 0.2)$

The document 1 vector entry for word w is:

$$\text{freqindoc}(w) \log_2(1 / \text{freq_in_bg}(w))$$

This is just a rephrasing of TFIDF, where: $\text{freqindoc}(w)$ is the frequency of w in document 1, and $\text{freq_in_bg}(w)$ is the ‘background’ frequency in our reference set of documents

Turning a document into a vector

Master list: (banana, cat, dog, fish, read)

Background frequencies: (0.2, 0.1, 0.05, 0.05, 0.2)

Document 1:

“Bananas are grown in hot countries, and cats like bananas.”

Frequencies are *proportions*. The background frequency of banana is 0.2, meaning that 20% of documents in general contain ‘banana’, or bananas, etc. (note that read includes reads, reading, reader, etc...)

The frequency of banana in document 1 is also 0.2 – why?

The TFIDF encoding of this document is:

0.464, 0.332, 0, 0, 0

Suppose another document has exactly the same vector – will it be the same document?

Jaccard's Coefficient of Link Neighbourhood

- The Jaccard Coefficient, also known as **Jaccard index** or **Jaccard similarity coefficient**, is a statistic measure used for comparing similarity of sample sets.
- It is usually denoted as $J(x, y)$ where x and y represent two different nodes in a network.
- In link prediction, all the neighbours of a node are treated as a set and the prediction is done by computing and ranking the similarity of the neighbour set of each node pair.
- This method is based on Common Neighbours method and its complexity is also $O(Nk^2)$.

Jaccard's Coefficient of Link Neighbourhood

- The mathematical expression of this method is as follows

- Score $(x, y) =$

$$\boxed{\frac{|\Gamma(x) \cap \Gamma(y)|}{|\Gamma(x) \cup \Gamma(y)|}}.$$

Jaccard's Coefficient of Link Neighbourhood

- Jaccard coefficient is one of the metrics used to compare the **similarity and diversity of sample sets**.
- It uses the ratio of the intersecting set to the union set as the measure of similarity.
- Thus it **equals to zero** if there are no intersecting elements and **equals to one** if all elements intersect.
Equation for Jaccard coefficient is

$$T = \frac{N_c}{N_a + N_b - N_c}$$

- where
- Na - number of elements in set A,
- Nb - number of elements in set B
- Nc - number of elements in intersecting set

Evaluation of Crawlers

- Goal: build “better” crawlers to support applications
- Build an unbiased evaluation framework
- Define common tasks of measurable difficulty
- Identify topics, relevant targets
- Identify appropriate performance measures
- **Effectiveness:** quality of crawler pages, order, etc.
- **Efficiency:** separate CPU & memory of crawler algorithms from bandwidth & common utilities
- *Perhaps the most crucial evaluation of a crawler is to measure the rate at which relevant web pages are acquired and how effectively irrelevant web pages are filtered out from the crawler.*

Evaluation of Crawlers

- With this knowledge, we could estimate the **precision and recall** of a crawler after crawling n web pages.
- The precision would be the fraction of pages crawled that are relevant to the topic and recall would be the fraction of relevant pages crawled.
- However, the relevant set for any given topic is unknown in the web, so the true recall is hard to measure.
- Therefore, we adopt **harvest rate and target recall** to evaluate the performance of the crawler.
- In case we have boolean relevance scores, we could measure the rate at which “good” pages are found; if 100 relevant pages are found in the first 500 pages crawled, we have an acquisition rate or **harvest rate** of 20% at 500 pages.

Evaluation of Crawlers

- The harvest rate is the fraction of web pages crawled that are relevant to the given topic, which measures how well it is doing at rejecting irrelevant web pages. The expression is given by:

$$\text{Harvest rate} = \frac{\sum_{i \in V} r_i}{|V|},$$

- where
- V is the number of web pages crawled by a crawler in current;
- r_i is the relevance between web page i and the given topic, and the value of r_i can only be 0 or 1.
- If relevant, then $r_i = 1$; otherwise $r_i = 0$.

Evaluation of Topical Crawlers

- The **target recall** is the fraction of relevant pages crawled, which measures how well it is doing at finding all the **relevant web pages**.
- However, the relevant set for any given topic is unknown in the Web, so the **true target recall is hard to measure**.
- In view of this situation, we delineate a specific network, where given a set of seed URLs and a certain depth, the range reached by a **crawler using breadth-first crawling strategy is the virtual Web**.
- We assume that the target set T is the relevant set in the virtual Web; C_t is the set of first t pages crawled. The expression is given by:

$$\text{Target recall} = \frac{|T \cap C_t|}{|T|}.$$

Evaluation of Topical Crawlers

Another measure from information retrieval that has been applied to crawler evaluation is **search length**,

- defined as the number of pages (or the number of irrelevant pages) crawled before a certain percentage of the relevant pages are found.
- Search length is a kind of the reciprocal of precision for a preset level of recall.

Evaluation of Topical Crawlers

- A second approach is to split the set of known relevant pages into two sets; **one set can be used as seeds, the other as targets.**
- While there is no guarantee that the **targets** are reachable from the **seeds**, this approach is significantly simpler because **no back-crawl** is necessary.
- Another advantage is that each of the two relevant subsets can be used in turn as **seeds and targets.**
- In this way, one can measure the overlap between the pages crawled starting from the two disjoint sets.

Crawler Etiquette

- **Identify yourself**
 - Use ‘User-Agent’ HTTP header to identify crawler, website with description of crawler and contact information for crawler developer
 - ❑ Use ‘From’ HTTP header to specify crawler developer email
 - ❑ Do not disguise crawler as a browser by using their ‘User-Agent’ string
- Always check that HTTP requests are successful, and in case of error, use HTTP error code to determine and immediately address problem
- Pay attention to anything that may lead to too many requests to any one server, even unwillingly, e.g.:
 - ✓ redirection loops
 - ✓ spider traps

Crawler Ettiquette : Server Aspects

- ***Spread the load, do not overwhelm a server***
- Make sure that no more than some max. number of requests to any single server per unit time, say < 1/second
- ***Honor the Robot Exclusion Protocol***
- A server can specify which parts of its document tree any crawler is or is not allowed to crawl by a file named ‘robots.txt’ placed in the HTTP root directory, **e.g.** <http://www.indiana.edu/robots.txt>
- Crawler should always check, parse, and obey this file before sending any requests to a server

Crawler Ethics Issues

- Is compliance with robot exclusion a matter of law?
- No! Compliance is voluntary, but if you do not comply, you may be blocked
- Someone (unsuccessfully) , Internet Archive over a robots.txt related issue
- Some crawlers disguise themselves
- Using false User-Agent
- Randomizing access frequency to look like a human/browser
- **Example: click fraud for ads**

Crawler Ethics Issues

- Servers can disguise themselves, too
- **Cloaking:** present different content based on UserAgent
 - E.g. stuff keywords on version of page shown to search engine crawler
- Search engines do not look kindly on this type of “spamdexing” and remove from their index sites that perform such abuse

WEB CRAWLING

Basic Crawler Algorithm: Breadth-First

POPULAR WEB CRAWLERS IN THE MARKET

Bot Name	% of Sites Crawled	Bot Type
Googlebot	96%	Search Bot
Baidu Spider	89%	Search Bot
MSN Bot/BingBot	82%	Search Bot
Yandex Bot	73%	Search Bot
Soso Spider	61%	Search Bot
ExaBot	35%	Search Bot
Sogou Spider	31%	Search Bot
Google Plus Share	24%	Crawler
Facebook External Hit	24%	Crawler
Google Feedfetcher	22%	Feed Fetcher

Web Crawler?

- The process or program used by search engines to download pages from the web for later processing by a search engine that will index the downloaded pages to provide fast searches
- A program or automated script which browses the World Wide Web in a methodical, automated manner
- Also known as web spiders and web robots
- Less used names- ants, bots and worms

Why web Crawlers?

- ❖ Internet has a wide expanse of Information
- ❖ **Finding relevant information requires an efficient mechanism**
- ❖ **Web Crawlers provide that scope to the search engine**

A screenshot of a Google search results page. The search bar at the top contains the query "vit". Below the search bar are navigation links for "All", "News", "Maps", "Images", "Videos", and "More". A red oval highlights the text "About 24,70,00,000 results (1.02 seconds)" which is displayed prominently. To the right of the search bar are "Settings" and "Tools" buttons. On the far right of the page is a small user profile icon.

About 24,70,00,000 results (1.02 seconds)

Ad · viteee.vit.ac.in/ 0416 220 2020

B. Tech Admissions Open In VIT - Online Application Form 2020

Ranked as one of the best engineering college in India for 3 years in a row by NIRF. World standard labs, e-Journals, technology business incubator at VIT. World ranked institution. Advanced teaching method. 300+ International MOU's. Best Placements. 700+ Recruiters. Programmes Offered · View Syllabus · Eligibility Criteria · Exam Pattern · Fees Structure

vit.ac.in ▾

VIT | No.1 Private Institution for Innovation

Established in 1984, VIT is a No.1 Progressive Educational Institution & Top Ranking University in India. Its a dedicated to the pursuit of excellence..

You've visited this page 2 times. Last visit: 27/8/19

VITEEE

Please visit only www.vit.ac.in or
<https://viteee.vit.ac.in> for ...

Admissions

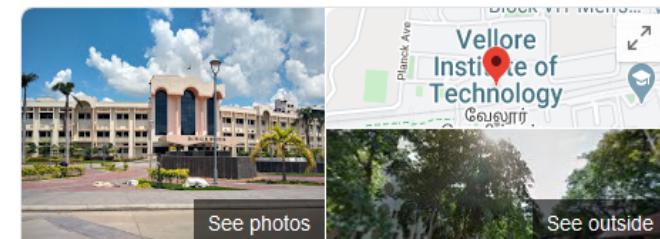
Postgraduate - Undergraduate

VIT | Chennai

Undergraduate - Admissions -
Programmes Offered - SAS - ...

Postgraduate

PG Application - Programmes



Vellore Institute of Technology

[Website](#) [Directions](#) [Save](#)

Private university in Vellore, Tamil Nadu

Vellore Institute of Technology is a private university located in Vellore, Tamil Nadu, India. Founded in 1984, as Vellore Engineering College, by G. Viswanathan, the institution offers 20 undergraduate, 34 postgraduate, four integrated and four research programs. [Wikipedia](#)

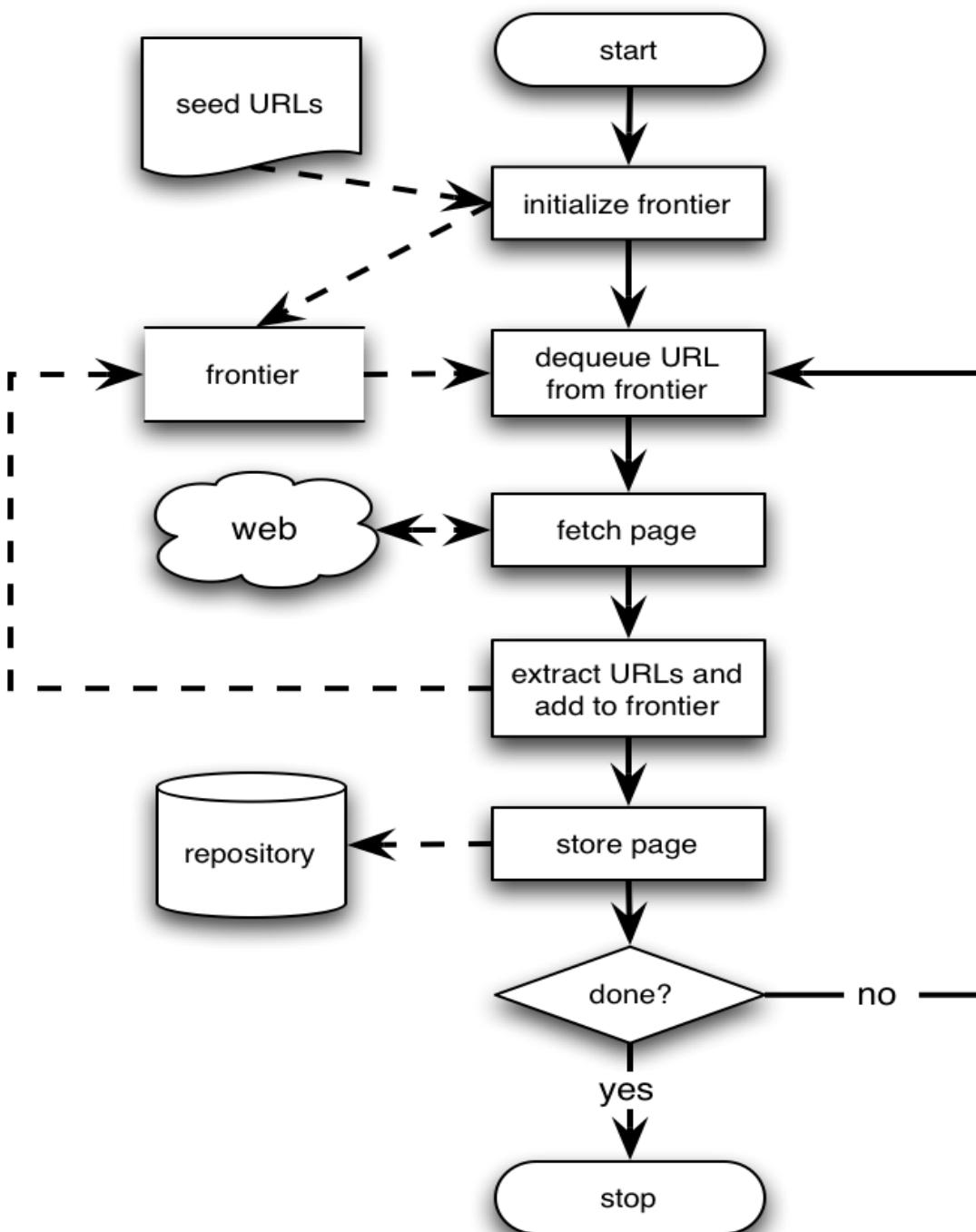
Crawl “all” Web pages?

- **Problem:** No catalog of all accessible URLs on the Web.
- **Solution:**
 - start from a given set of URLs
 - Progressively fetch and scan them for new outlinking URLs
 - fetch these pages in turn
 - Submit the text in page to a text indexing system

How does web crawler work?

- It starts with a list of URLs to visit, called the **seeds..**
- As the crawler visits these URLs, it identifies all the hyperlinks in the page and adds them to the list of visited URLs, called the **crawl frontier.**
- URLs from the frontier are recursively visited according to a **set of policies.**

Basic crawlers



- This is a **sequential** crawler
- **Seeds** can be any list of starting URLs
- Order of page visits is determined by **frontier** data structure
- **Stop** criterion can be anything

Crawling procedure

- **Simple**
 - Great deal of engineering goes into industry-strength crawlers
 - Industry crawlers crawl a substantial fraction of the Web
 - E.g.: Alta Vista, Northern Lights, HTTtracker
- No guarantee that all accessible Web pages will be located in this fashion
- **Crawler may never halt**
 - pages will be added continually even as it is running.

Crawling overheads

- Delays involved in
 - Resolving the host name in the URL to an IP address using DNS
 - Connecting a socket to the server and sending the request
 - Receiving the requested page in response
- Solution: Overlap the above delays by
 - fetching many pages at the same time

Anatomy of a crawler

- **Page fetching threads**
 - Starts with DNS resolution
 - Finishes when the entire page has been fetched
- **Each page**
 - stored in compressed form to disk/tape
 - scanned for outlinks
- **Work pool of outlinks**
 - maintain network utilization without overloading it
 - Dealt with by *load manager*
- Continue till the crawler has collected a *sufficient* number of pages.

Applications of Web Crawlers

- **Business Intelligence**

Collect information about their competitors and potential collaborators

- **Monitor Web site and Pages of Interest**

A user or community can be notified when new information appears in certain places

- **Support of Search Engine**

Crawlers are the main consumers of Internet bandwidth.

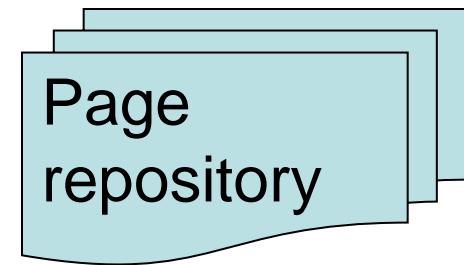
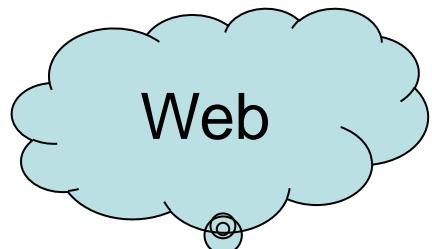
They collect pages for search engines to build their indexes.

Applications of Web Crawlers

Harvest Email-Address (Malicious Application)

- Used by spammers or collect personal information to be used in **phishing and other identity theft attacks**.
- Well known search engines such as Google, *Yahoo!* and MSN run very efficient **universal crawlers** designed to gather all pages irrespective of their content.
- Other crawlers, sometimes called **preferential crawlers**, are more targeted.
 - They attempt to download only pages of certain **types or topics**.

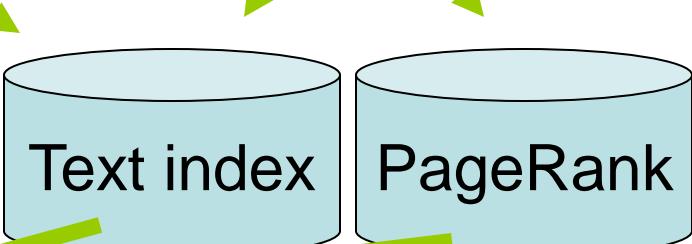
A crawler within a search engine



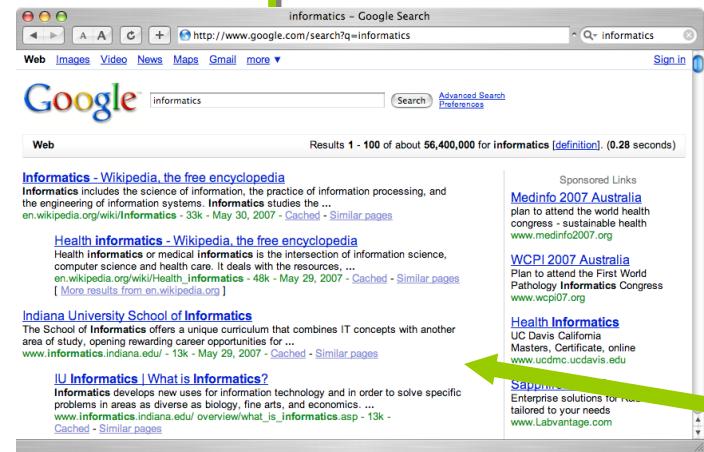
Query

Text & link
analysis

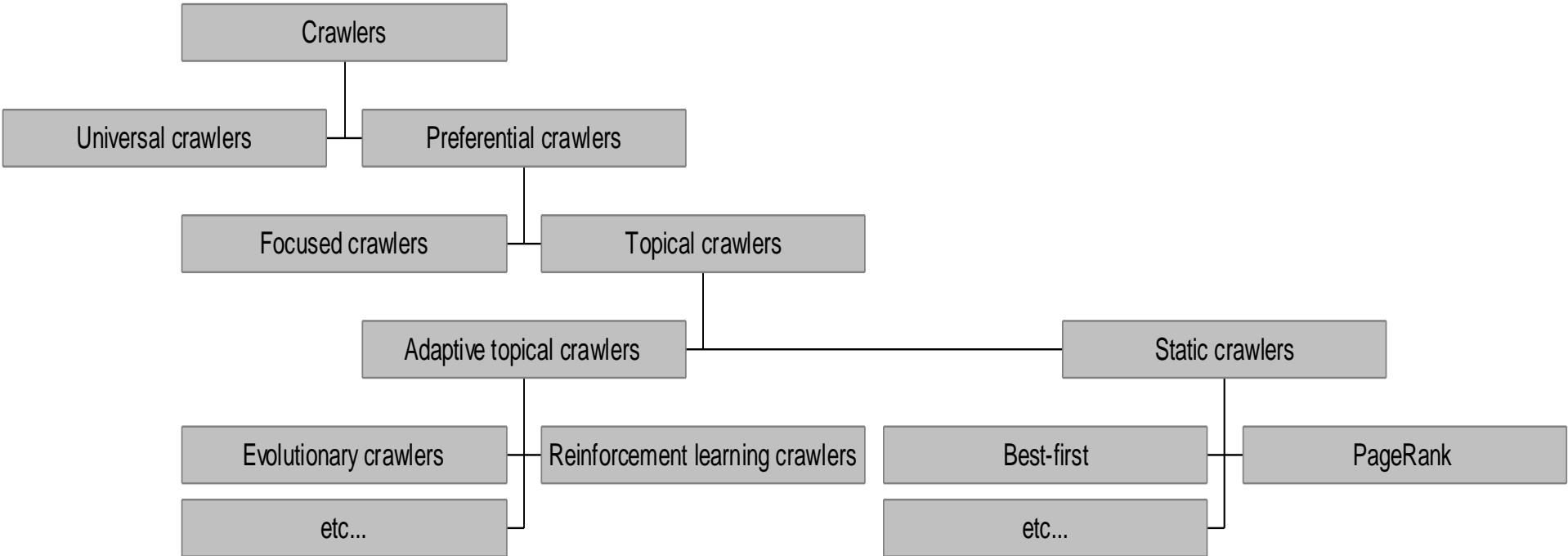
hits



Ranker

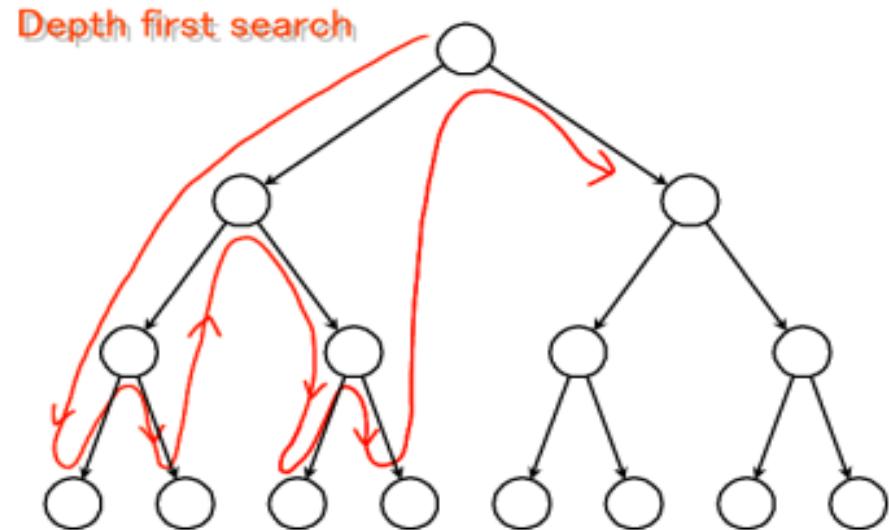
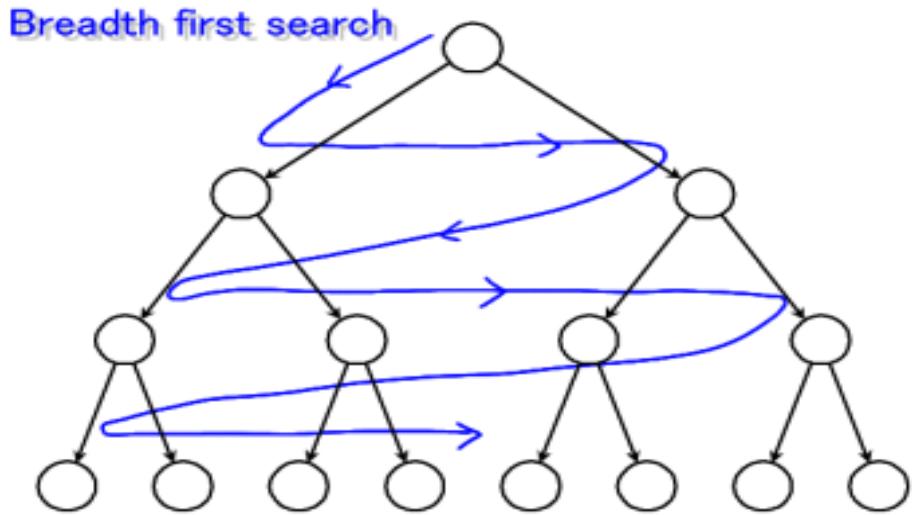


One taxonomy of crawlers

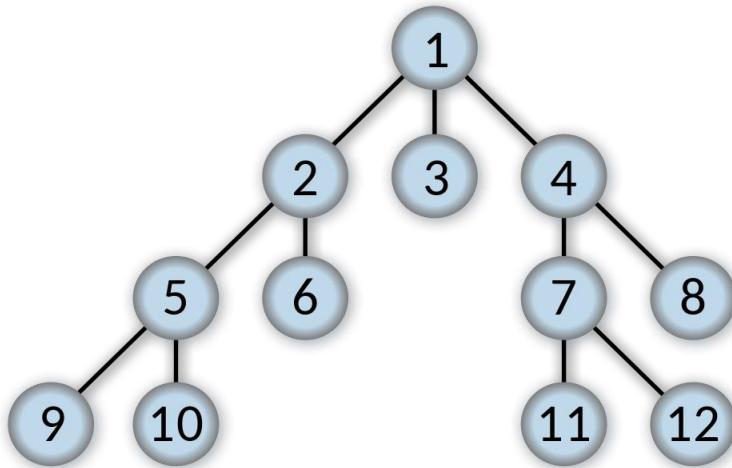


Graph traversal (BFS or DFS?)

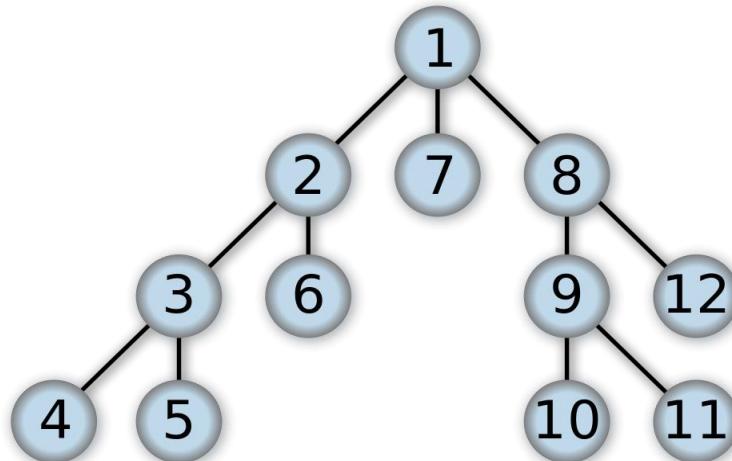
- Breadth First Search
 - Implemented with QUEUE (FIFO)
 - Finds pages along shortest paths
 - If we start with “good” pages, this keeps us close; maybe other good stuff
- Depth First Search
 - Implemented with STACK (LIFO)
 - Wander away (“lost in cyberspace”)



BFS vs DFS



Breadth-first search



Depth-first search

Breadth-First Crawlers

Breadth-First Crawlers

- The frontier may be implemented as a first-in-first-out (FIFO) queue, corresponding to a **breadth-first** crawler.
- The URL to crawl next comes from the **head of the queue** and new URLs are **added** to the **tail of the queue**.
- Once the **frontier reaches its maximum size**, the breadth-first crawler can add to the queue only one unvisited URL from each new page crawled.

Breadth-First Crawlers

- The breadth-first strategy does not imply that pages are visited in “random” order. Because
- The order in which pages are visited by a breadth-first crawler is highly correlated with their **PageRank or indegree values**.
- They are greatly affected by the choice of **seed pages**.
- Topical locality measures indicate that pages in the link neighborhood of a seed page are much more likely to be related to the seed pages than randomly selected pages.

Crawling Algorithm

Initialize queue (Q) with initial set of known URL's.

Until Q empty or page or time limit exhausted:

Pop URL, L, from front of Q.

If L is not an HTML page (.gif, .jpeg, .ps, .pdf, .ppt...)
exit loop.

If already visited L, continue loop(get next url).

Download page, P, for L.

If cannot download P (e.g. 404 error, robot excluded)
exit loop, else.

Index P (e.g. add to inverted index or store cached copy).

Parse P to obtain list of new links N.

Append N to the end of Q.

Implementation issues

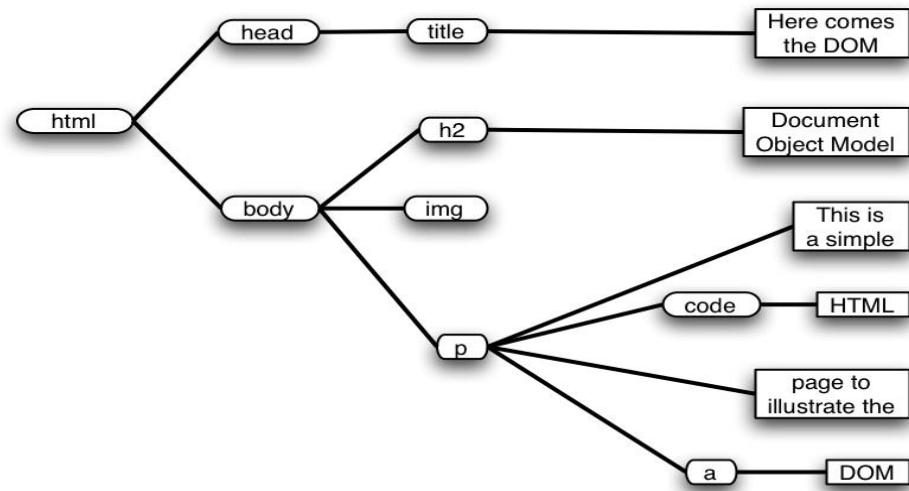
- Don't want to fetch same page twice!
 - Keep lookup table (hash) of visited pages
 - What if not visited but in frontier already?
- The frontier grows very fast!
 - May need to prioritize for large crawls
- Fetcher must be robust!
 - Don't crash if download fails
 - Timeout mechanism
- Determine file type to skip unwanted files
 - Can try using extensions, **but not reliable**
 - Can issue 'HEAD' HTTP commands to get Content-Type (MIME) headers, but overhead of extra Internet requests

More implementation issues

- **Fetching**
 - Get only the first 10-100 KB per page
 - Take care to detect and break redirection loops
 - **Soft fail** for timeout, server not responding, file not found, and other errors

More implementation issues: Parsing

- HTML has the structure of a **DOM (Document Object Model)** tree
- Unfortunately actual HTML is often incorrect in a strict syntactic sense
- Crawlers, like browsers, must be **robust/forgiving**
- Must pay attention to HTML entities and unicode in text
- What to do with a growing number of other formats?
 - Flash, SVG, RSS, AJAX...



More implementation issues

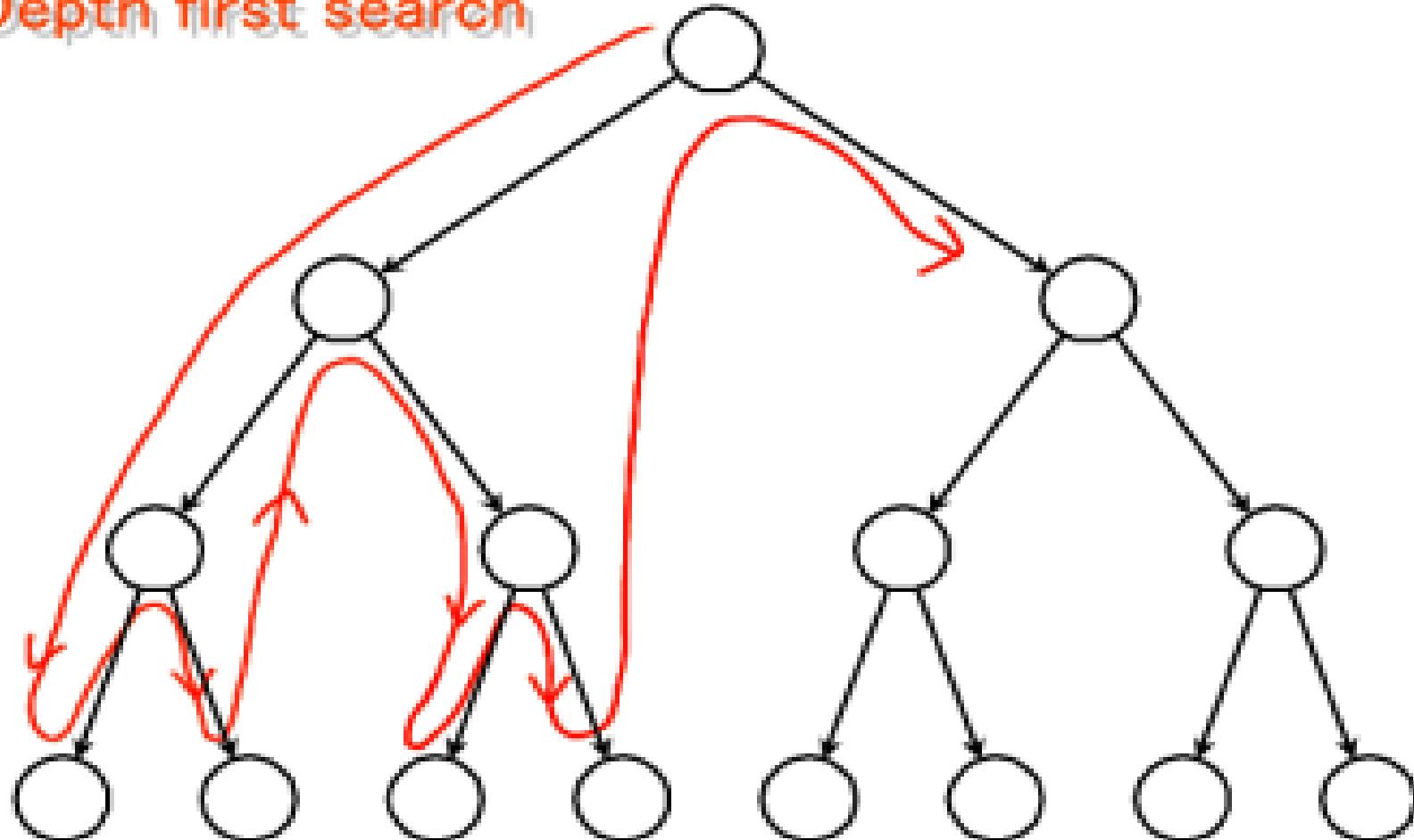
- Stop words
 - Noise words that do not carry meaning should be eliminated (“stopped”) before they are indexed
 - E.g. in English: AND, THE, A, AT, OR, ON, FOR, etc...
 - Typically syntactic markers
 - Typically the most common terms
 - Typically kept in a negative dictionary
 - 10–1,000 elements
 - E.g. http://ir.dcs.gla.ac.uk/resources/linguistic_utils/stop_words
 - **Parser** can detect these right away and disregard them

Depth-First Crawlers

- The frontier may be implemented as a last-in-first-out (LIFO) stack, corresponding to a **depth-first** crawler.
- Wander away (“lost in cyberspace”) Use depth first search (DFS) algorithm
- Get the 1st link not visited from the start page
- Visit link and get 1st non-visited link
- Repeat above step till no non-visited links
- Go to next non-visited link in the previous level and repeat 2nd step

Depth-First Crawlers

Depth first search



Breadth-First vs. Depth-First Crawlers

- breadth-first is more careful by checking all alternatives
- complete and optimal
- very memory-intensive
- depth-first goes off into one branch until it reaches a leaf node
- not good if the goal node is on another branch
- neither complete nor optimal
- uses much less space than breadth-first

WEB MINING

Preferential Crawlers

Preferential Crawlers

- A different crawling strategy is obtained if the frontier is implemented as a **priority queue** rather than a FIFO queue.
- **Preferential crawlers** assign each unvisited link a priority based on an estimate of the value of the **linked page**.
- **The estimate can be based on**
 - Topological properties
(e.g., the indegree of the target page)
 - Content properties (e.g., the similarity between a user query and the source page)
 - Combination of measurable features.

Preferential Crawlers

- If pages are visited in the order specified by the **priority values** in the frontier, then we have a **best-first** crawler.
- Best First Search is a **heuristic** based search algorithm.
- In this approach, **relevancy calculation** is done for **each link** and the most relevant link, such as one with the highest relevancy value, is fetched from the frontier.
- Thus every time the best available link is **opened and traversed**.

Preferential Crawlers

- The **time complexity** of inserting a URL into the priority queue is $O(\log F)$, where F is the frontier size (looking up the hash requires constant time).
- To dequeue a URL, it must first be removed from the priority queue ($O(\log F)$) and then from the **hash table** (again $O(1)$).
- Thus the parallel use of the two data structures yields a logarithmic total **cost per URL**.
- Once the frontier's **maximum size** is reached, only the best URLs are kept; the frontier must be pruned after each new set of links is added.

Preferential crawling algorithms: Examples

- **Breadth-First**
 - Exhaustively visit all links in order encountered
- **Best-N-First**
 - Priority queue sorted by similarity, explore top N at a time
 - Variants: DOM context
- **PageRank**
 - Priority queue sorted by keywords, PageRank
- **SharkSearch**
 - Priority queue sorted by combination of similarity, anchor text, similarity of parent, etc. (powerful cousin of FishSearch)
- **InfoSpiders**
 - Adaptive distributed algorithm using an evolving population of learning agents

Fish Search

- Fish Search is a dynamic heuristic search algorithm.
- It works on the intuition that **relevant links** have **relevant neighbours**;
- hence it **starts** with a **relevant link** and goes deep under that link and stops searching under the links that are irrelevant.
- The key point of Fish Search algorithm lies in the maintenance of **URL order**.

A* Search

- A* uses Best First Search.
- It calculates the relevancy of each link and the difference between expected relevancy of the target web-page and the current link.
- The total of these two values serve as the measure for selecting the best path.

Adaptive A* Search

- Adaptive A* Search works on **informed heuristics** to focus its searches.
- With each **iteration**, it updates the **relevancy value** of the page and uses it for the **next traversal**.
- The pages are updated for **$\log(\text{Graph Size})$** times, (**$\log(\text{Graph Size})$** times the overhead of updating is much more than the improvement that can be achieved in getting more relevant pages).

More implementation issues

Conflation and thesauri

- Idea: improve **recall** by merging words with **same meaning**
 1. We want to ignore superficial **morphological** features, thus merge semantically similar tokens
 - {student, study, studying, studious} => studi
 2. We can also conflate **synonyms** into a single form using a thesaurus
 - 30-50% smaller index
 - Doing this in both pages and queries allows to retrieve pages about ‘automobile’ when user asks for ‘car’
 - Thesaurus can be implemented as a hash table

More implementation issues

- **Stemming**
 - In linguistic morphology and information retrieval, stemming is the process of **reducing inflected** (or sometimes derived) words to **their word stem, base or root form.**
 - A stemming algorithm might also reduce the words *fish**ing*, *fishe**d*, and *fisher* to the stem ***fish***.

More implementation issues

- **Static vs. dynamic pages**

- Is it worth trying to eliminate dynamic pages and only index static pages?
- Examples:
 - <http://www.census.gov/cgi-bin/gazetteer>
 - <http://informatics.indiana.edu/research/colloquia.asp>
 - <http://www.amazon.com/exec/obidos/subst/home/home.html/02-8332429-6490452>
 - <http://www.imdb.com/Name?Menczer,+Erico>
 - <http://www.imdb.com/name/nm0578801/>
- Why or why not? How can we tell if a page is dynamic?
What about '**spider traps**'?
- What do Google and other search engines do?

More implementation issues

- **Relative vs. Absolute URLs**

- Crawler must translate relative URLs into absolute URLs
- Need to obtain Base URL from HTTP header, or HTML Meta tag, or else current page path by default
- Examples
 - **Base:** `http://www.vit.ac.in/schools.html`
 - **Relative URL:** `schools.html`
 - **Absolute URL:** `https://www.vit.ac.in/schools.html`

More implementation issues

- **URL canonicalization**
 - All of these:
 - <http://www.cnn.com/TECH>
 - <http://WWW.CNN.COM/TECH/>
 - <http://www.cnn.com:80/TECH/>
 - <http://www.cnn.com/bogus/.../TECH/>
 - Are really equivalent to this canonical form:
 - <http://www.cnn.com/TECH/>
 - In order to avoid duplication, the crawler must transform all URLs into canonical form
 - Definition of “canonical” is arbitrary, e.g.:
 - Could always include port
 - only include port when not default :80

More implementation issues

- Spider traps
 - **Misleading sites:** indefinite number of pages dynamically generated by **CGI scripts**
 - Paths of arbitrary depth created using soft directory links and path rewriting features in HTTP server
 - Only heuristic defensive measures:
 - Check URL length; assume spider trap above some threshold, for example 128 characters
 - Watch for sites with very large number of URLs
 - Eliminate URLs with non-textual data types
 - May disable crawling of dynamic pages, if can detect

More implementation issues

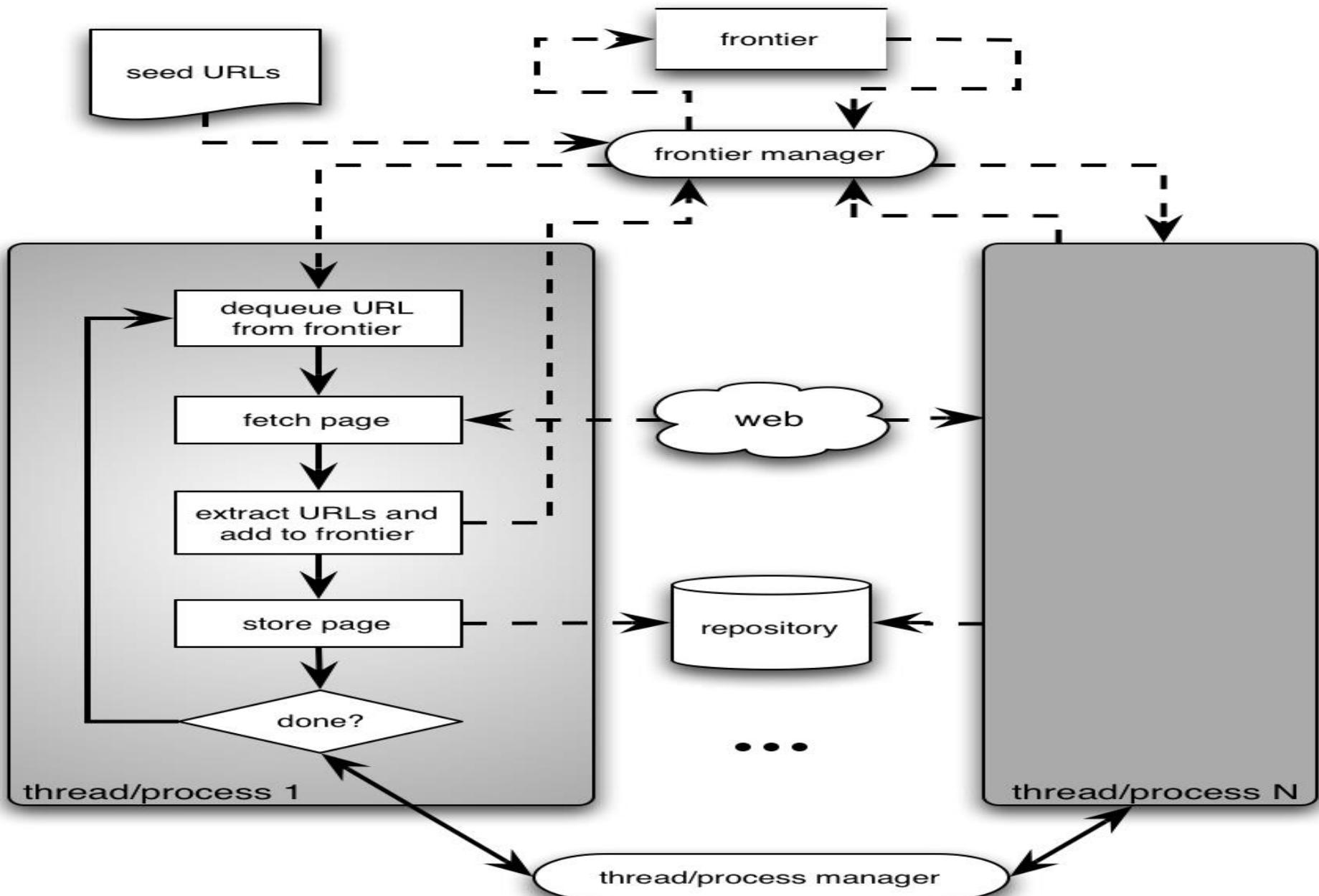
- **Page repository**

- Naïve: store each page as a separate file
 - Can map URL to unique filename using a hashing function
 - This generates a huge number of files, which is inefficient from the storage perspective
- Better: combine many pages into a single large file, using some XML markup to separate and identify them
 - Must map URL to {filename, page_id}
- Database options
 - Any RDBMS -- large overhead
 - Light-weight, embedded databases

Concurrency

- A crawler incurs several delays:
 - Resolving the host name in the URL to an IP address using DNS
 - Connecting a socket to the server and sending the request
 - Receiving the requested page in response
- Solution: Overlap the above delays by **fetching many pages concurrently**

Architecture of a concurrent crawler



Concurrent crawlers

- Can use **multi-processing** or **multi-threading**
- Each process or thread works like a sequential crawler, except they **share data structures**: frontier and repository
- Shared data structures must be synchronized (locked for concurrent writes)
- Speedup of factor of 5-10 are easy this way

WEB MINING

Universal Crawlers – Focused Crawlers

What is Universal Crawler ?

- Universal crawler is a **fast** precise and **reliable** Internet crawler.
- Methodical, automated manner and creates the index of documents

How UC works?

- Uc crawler Downloads first website.
- It goes through html, finds an **link tag**, and retrieves **outside link**.
- When it finds "`<ahref="http://www.sourceforge.net"> website `" it copies the link and adds it to a **list of pages it plans to crawl**.
- It crawls it and analyzes it again
- Each website is saved into "web" folder, so there is **no need** to **re-download pages** if job was stopped.

Universal crawlers : Basics

- Support universal search engines
- Large-scale
- Huge cost (network bandwidth) of crawl is amortized over many queries from users
- **Incremental updates** to existing index and other data repositories

Large-scale universal crawlers

Two major issues:

1. Performance

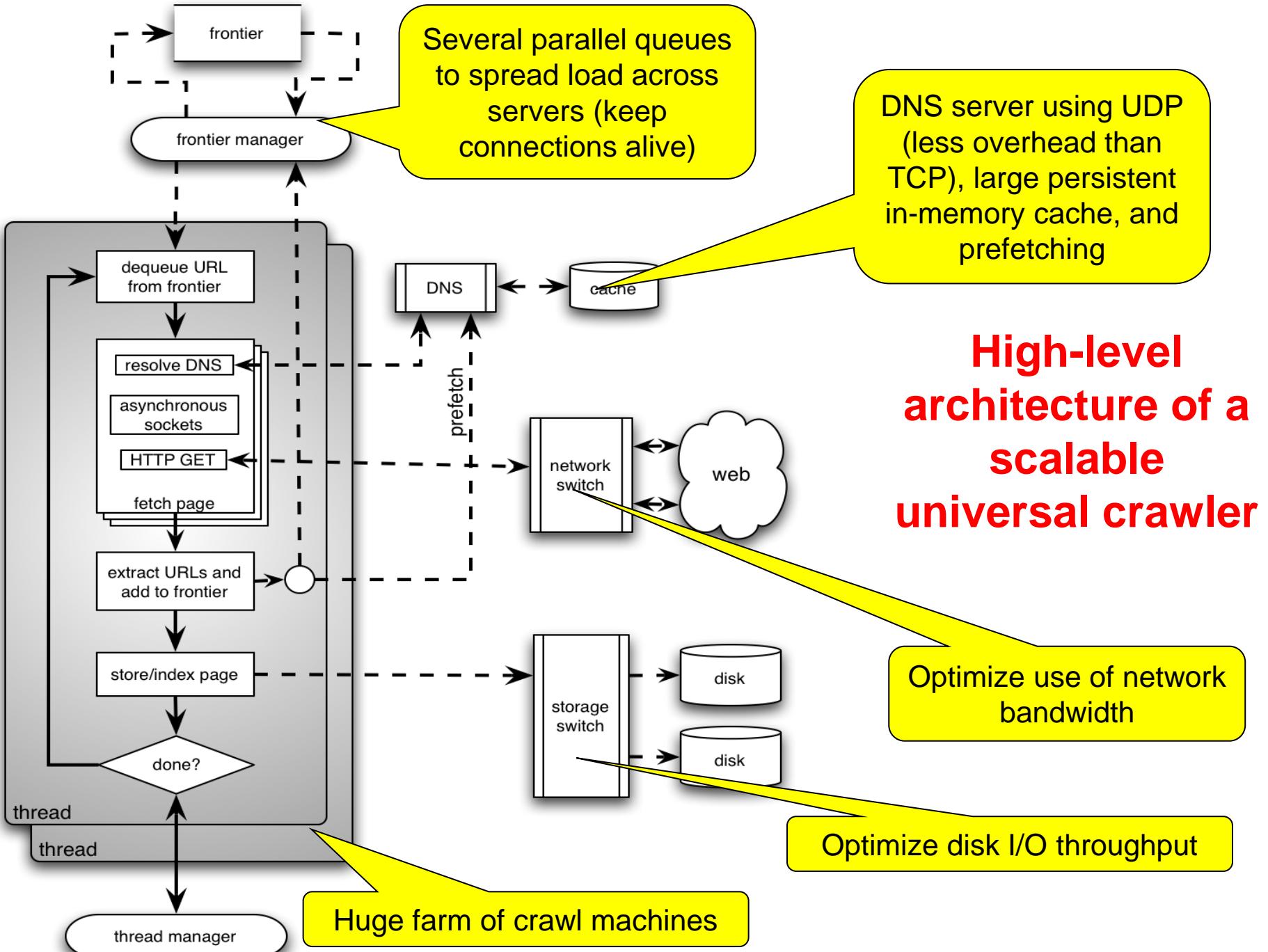
- Need to scale up to billions of pages

2. Policy

- Need to trade-off coverage, freshness, and bias (e.g. toward “important” pages)

Large-scale crawlers: scalability

- Need to minimize overhead of **DNS lookups**
- Need to optimize utilization of network bandwidth and disk throughput (I/O is bottleneck)
- Use **asynchronous sockets**
 - **Multi-processing or multi-threading** do not scale up to billions of pages
 - **Non-blocking:** hundreds of network connections open simultaneously
 - **Polling socket** to monitor completion of network transfers



Universal crawlers: Policy

- Coverage
 - New pages get added all the time
 - Can the crawler find every page? Yes
- Freshness
 - Pages change over time, get removed, etc.
 - How frequently can a crawler revisit ? Timeout
- Trade-off!
 - Focus on most “important” pages (crawler bias)?
 - “Importance” is subjective

Estimating page change rates

- Algorithms for maintaining a crawl in which most pages are fresher than a specified.
- Assumption: predicts the future
 - Frequency of change not a good predictor
 - Degree of change is a better predictor

Do we need to crawl the entire Web?

- If we cover too much, it will get **stale**
- There is an abundance of pages in the Web
- For PageRank, pages with very low prestige are **largely useless**
- **What is the goal?**
 - *General search engines: pages with **high prestige***
 - *News portals: pages that **change often***
 - *Vertical portals: pages on some topic*

Benefits of UC

- Get local copy of crawled pages.
- Get link analysis on crawled pages using Page Rank.
- Resume download only on pages not crawled

FOCUSED CRAWLERS

Focussed Crawlers

- Crawl only pages in **certain categories**
- A focused Crawler attempts to bias the crawler towards pages in some categories in which user is interested.
- One application of a such preferential crawler would be to maintain a **web taxonomy** such as Yahoo! Directory (dir.yahoo.com) or the volunteer based Open Directory Project (ODP, demoz.org).

Functionality of the Classifier

- The **classifier** would guide the crawler by preferentially selecting from **frontier pages** that appear most likely to belong to categories of interest, according to **classifier's prediction**.

Focused crawler Components

The focused crawler has three main components:

- 1. Classifier**
- 2. Distiller**
- 3. Crawler**

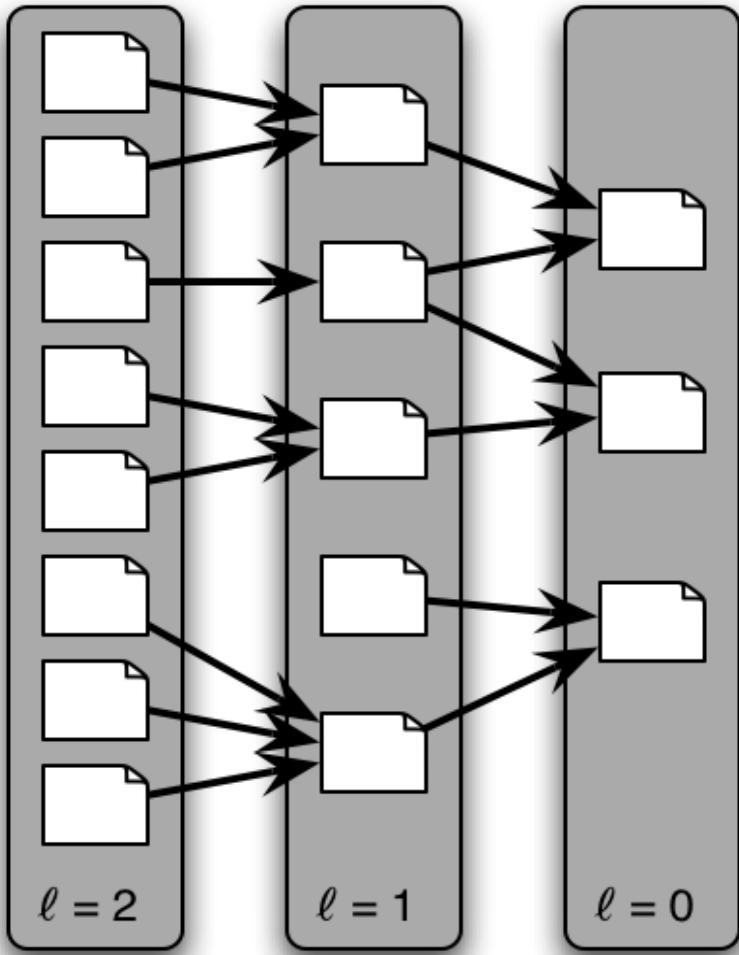
Focused crawlers

- Can have **multiple topics** with as many classifiers, with scores appropriately combined
- Can use a **distiller** to find topical hubs periodically, and add these to the frontier
- Can use alternative classifier algorithms to naïve-Bayes, e.g. **SVM** and **neuralnets** have reportedly performed better

Soft Focused Strategy & Hard Focused Strategy

- **Soft Focused Strategy** : the crawler uses the score $R(p)$ of each crawled page p as a **priority value** of each crawled page p as a priority value for all unvisited URLs extracted from p . The URLs are added to the frontier.
- **Hard Focused Strategy** : For a crawled page p , the classifier first finds the **leaf category** $C(p)$ in the taxonomy most likely to include p . If an ancestor of $C(p)$ is a focus category ,then URLs from the crawled page p are added to the frontier otherwise they are discarded.

Context-focused crawlers



Context graph

- Same idea, but **multiple classes** (and classifiers) based on **link distance** from relevant targets
 - $\ell=0$ is topic of interest
 - $\ell=1$ link to topic of interest
 - Etc.
- Initially needs a back-crawl from seeds (or known targets) to train classifiers to estimate distance
- Links in frontier prioritized based on estimated distance from targets
- Outperforms standard focused crawler empirically

TOPICAL CRAWLERS

Topical crawlers : Introduction

- A way to address the scalability limitations of universal search engines, by distributing the crawling process across **users, queries or even client computers.**
- Can guide the **navigation of links** with the goal of efficiently locating highly **relevant target pages**

Topical crawlers

- It is a topic (query, description, keywords) and a set of seed pages (not necessarily relevant)
- No labeled examples
- Must predict relevance of **unvisited links** to **prioritize**
- Original idea: Menczer 1997, Menczer & Belew 1998

Topical Crawler

- Example:-**MySpiders applet** (myspiders.informatics.indiana.edu). The applet is designed to demonstrate two topical crawling algorithms, **best-N-first** and **InfoSpiders**
- Unlike a search engine, this application has **no index to search for results**.
- Instead the Web is crawled in real time. As pages deemed relevant are crawled, they are displayed in a list that is kept sorted by a user-selected criterion: **score or recency**.
- The **score** is simply the content (cosine) similarity between a page and the query, and the **recency** of a page is estimated by the last-modified header, if returned by the server (not a very reliable estimate).

Pros and Cons of Topical Crawler

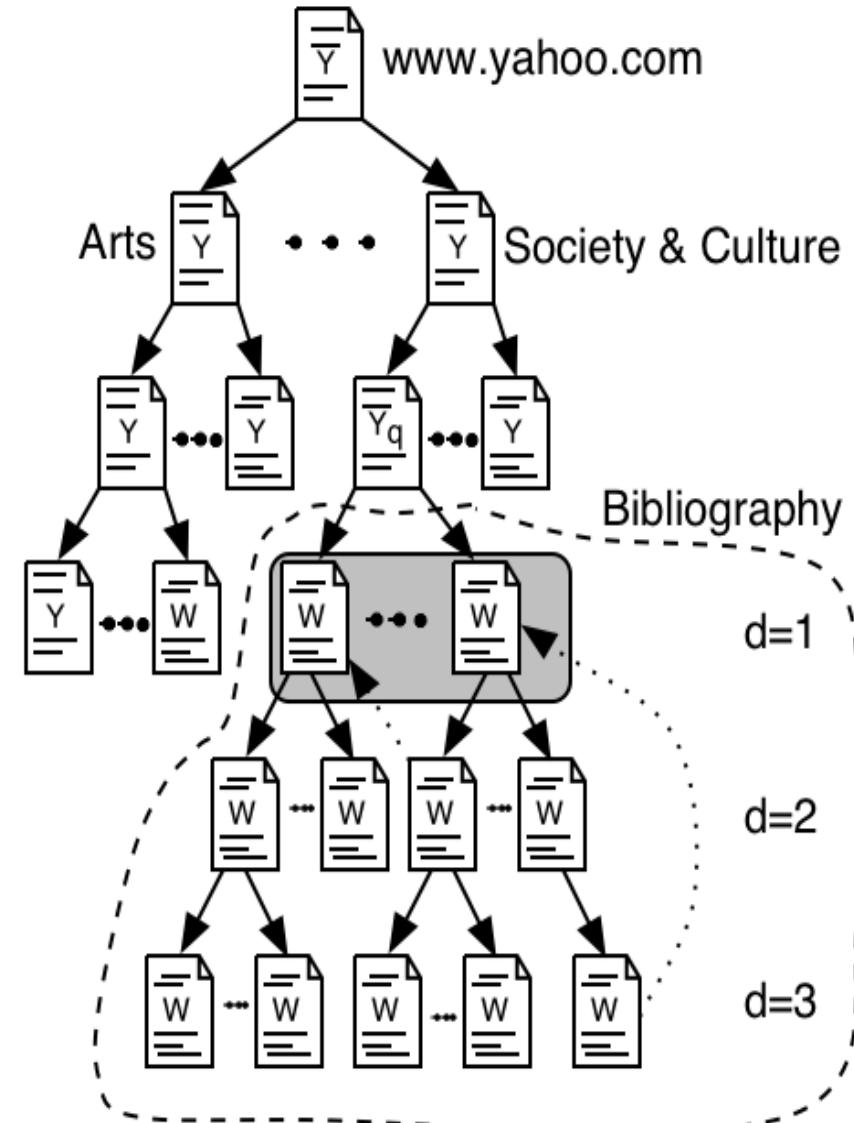
- All **hits** are fresh by definition.
- This makes this type of crawlers suitable for applications that look for very **recently posted documents**, which a search engine may not have indexed yet.
- On the ***down side***, the search is **slow compared** to a traditional search engine because the user has to wait while the **crawler fetches and analyzes pages**.
- Another disadvantage is that the **ranking algorithms** cannot take advantage of global **prestige measures**, such as **PageRank**, available to a traditional search engine.

Topical locality

- Topical locality is a **necessary** condition for a topical crawler to work, and for surfing to be a worthwhile activity for humans
- Links must encode **semantic** information, i.e. say something about neighbor pages, not be random
- It is also a **sufficient** condition if we start from “good” seed pages
- Crawling algorithms can use words and hyperlinks, associated respectively with a **lexical and a link topology**.
- In the former, **two pages** are close to each other if they have **similar textual content**; in the latter, if there is a **short path between them**

Quantifying topical locality

- Different ways to pose the question:
 - How quickly does semantic **locality** decay?
 - How fast is topic drift?
 - How quickly does **content change** as we surf away from a starting page?
- To answer these questions, let us consider **exhaustive** crawls
TECHNIQUES



Topical locality

- Link-content conjecture
- link-cluster conjecture:

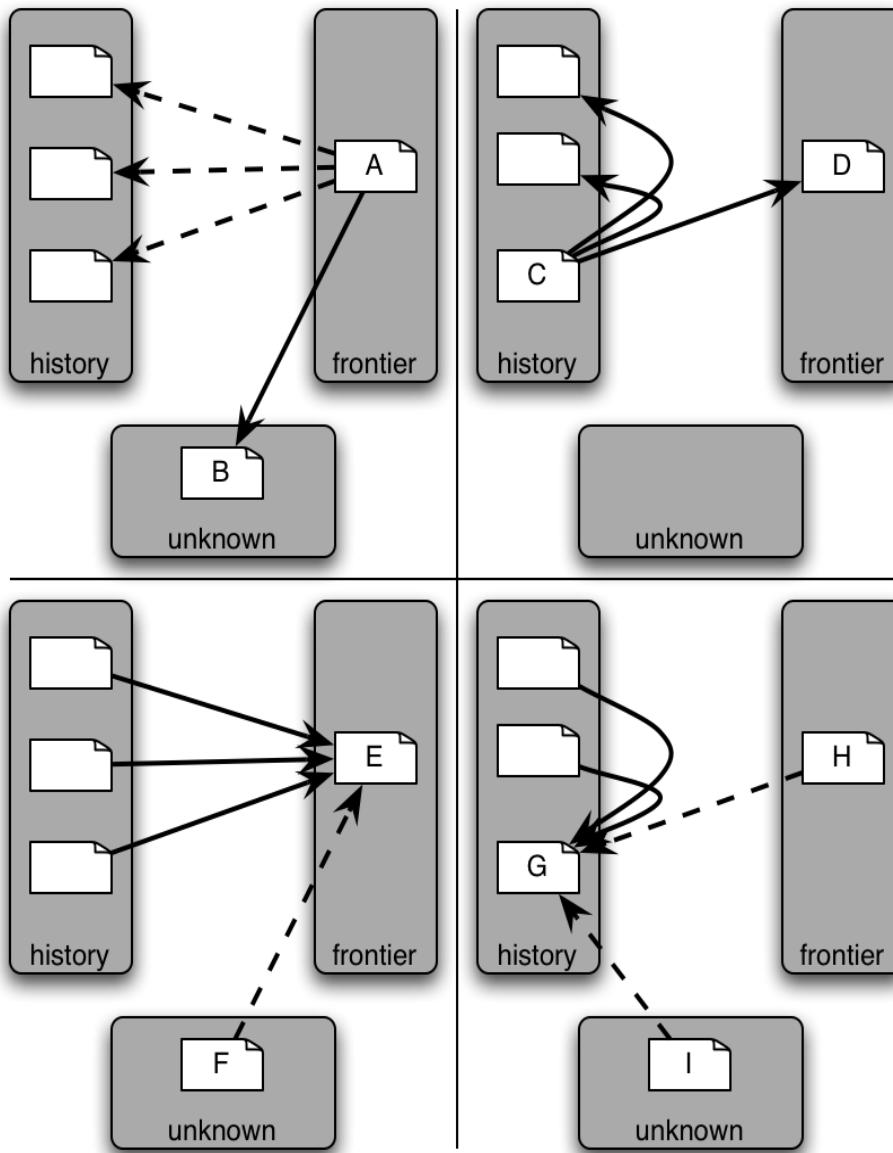
Link-content conjecture

- whether two pages that link to each other are more likely to be **lexically** similar to each other, compared to two **randomly selected pages**.
- This makes the assumption that pages which link to each other are closely **topically related**,
- *i.e. a page on bananas is likely to link to other pages about bananas (or at least fruit).*

Link-cluster conjecture

- **link-cluster conjecture:** whether two pages that link to each other are more likely to be semantically related to each other, compared to two randomly selected pages.
- This assumes that pages which are clustered together, or in the same "**web-community**" are closely topically related, *i.e. the page about bananas from above links to a page about fruit which links to a page about food - all three are closely topically related.*

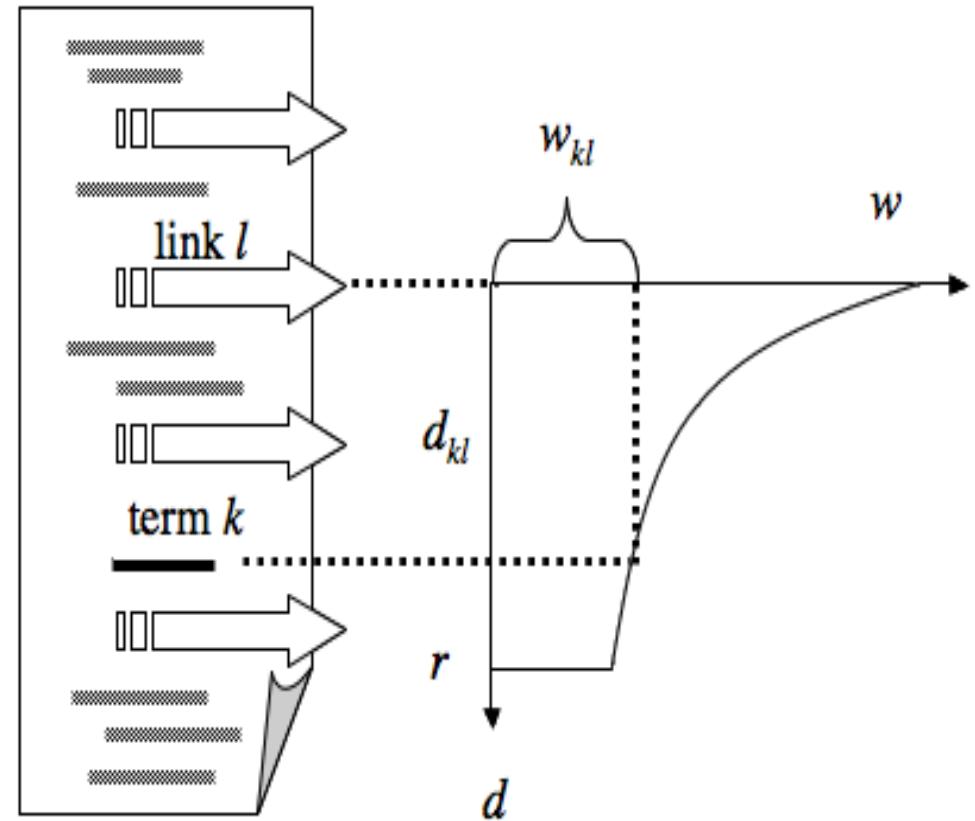
Topical locality-inspired for topical crawlers



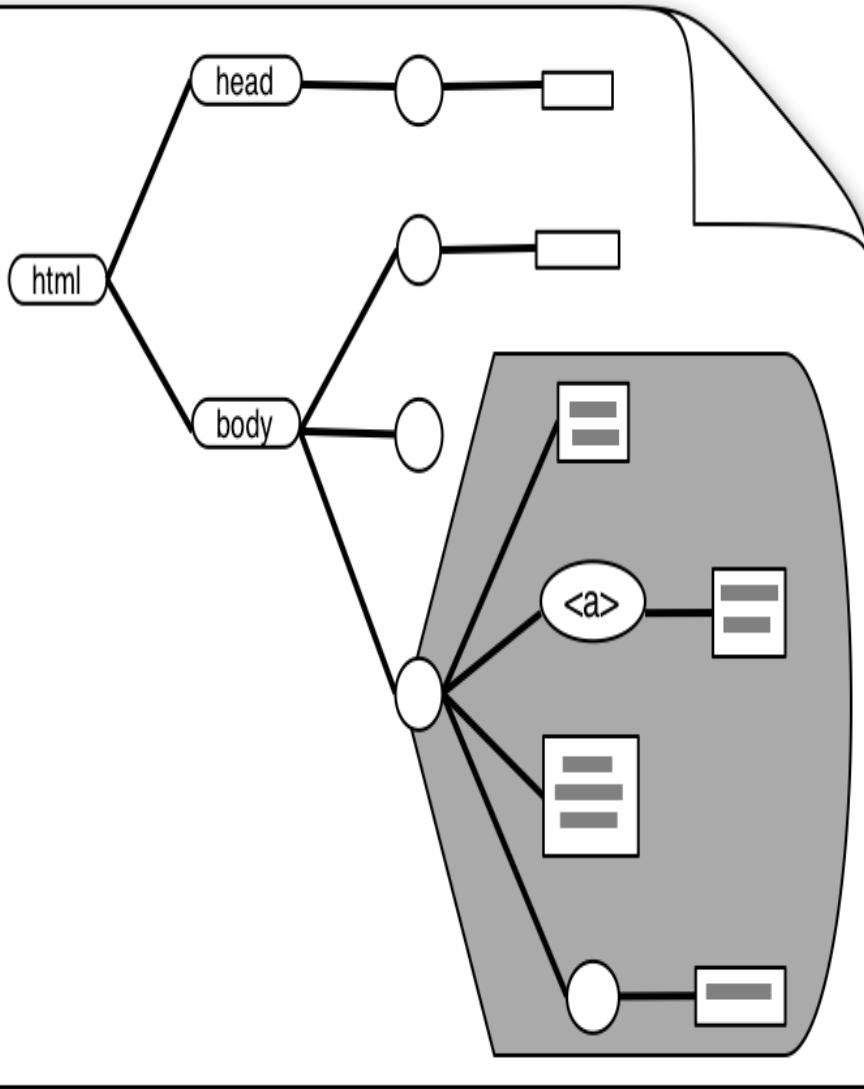
- Co-citation (a.k.a. **sibling locality**): A and C are good **hubs**, thus A and D should be given **high priority**
- Co-reference (a.k.a. **bibliographic coupling**): E and G are good **authorities**, thus E and H should be given **high priority**

Link context based on text neighborhood

- Often consider a fixed-size window, e.g. 50 words around anchor
- Can weigh links based on their distance from topic keywords within the document ?
- **YES, (*InfoSpiders, Clever*)**
- **Anchor text deserves extra importance**



Link context based on DOM tree



- Consider DOM subtree rooted at parent node of link's `<a>` tag
- Or can go further up in the tree (Naïve Best-First is special case of entire document body)

About Exelixis

Exelixis, Inc. is a leading genomics-based drug discovery company focused on product development through its expertise in comparative genomics and model system genetics. These technologies provide a rapid, efficient and cost effective way to move from DNA sequence data to knowledge about the function of genes and the proteins they encode. The company's technology is broadly applicable to all life sciences industries including pharmaceutical, diagnostic, agricultural biotechnology and animal health. Exelixis has partnerships with Aventis CropScience S.A., Bayer Corporation, Bristol-Myers Squibb Company, Elan Pharmaceuticals, Inc., Pharmacia Corporation, Protein Design Labs, Inc., Scios Inc. and Dow AgroSciences LLC, and is building its internal development program in the area of oncology. For more information, please visit the company's web site at www.exelixis.com.

<P class=MsoNormal>
About Exelixis
Exelixis, Inc. is a leading genomics-based drug discovery company focused on product development through its expertise in comparative genomics and model system genetics. These technologies provide a rapid, efficient and cost effective way to move from DNA sequence data to knowledge about the function of genes and the proteins they encode. The company's technology is broadly applicable to all life sciences industries including pharmaceutical, diagnostic, agricultural biotechnology and animal health. Exelixis has partnerships with Aventis CropScience S.A., Bayer Corporation, Bristol-Myers Squibb Company, Elan Pharmaceuticals, Inc., Pharmacia Corporation, Protein Design Labs, Inc., Scios Inc. and Dow AgroSciences LLC, and is building its internal development program in the area of oncology. For more information, please visit the company's web site at
www.exelixis.com.<o:p></o:p>
</P>

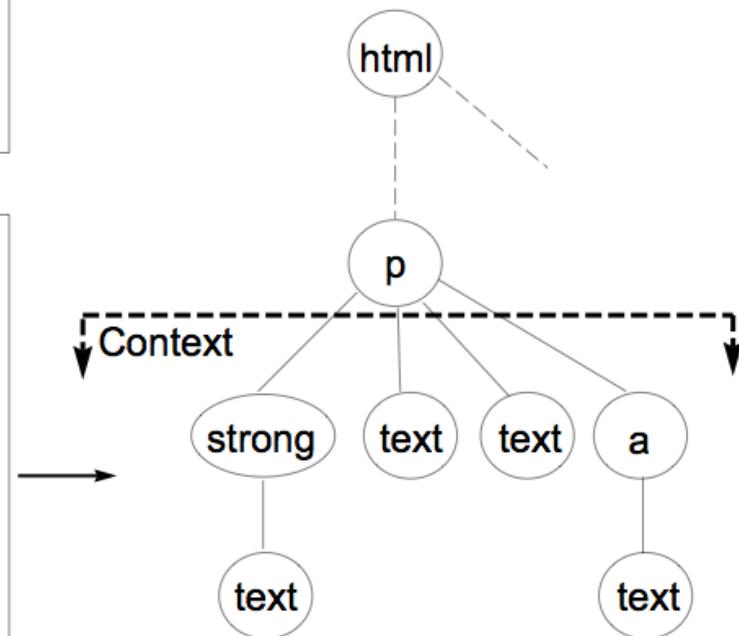
- <p>
-
 <text>about exelixis</text>

<text>exelixis inc is a leading genomics based drug discovery company focused on product development through its expertise in comparative genomics and model system genetics these technologies provide a rapid efficient and cost effective way to move from dna sequence data to knowledge about the function of genes andthe proteins they encode the company s technology is broadly applicable to all life sciences industries including pharmaceutical diagnostic agricultural biotechnology and animal health exelixis has partnerships with aventis cropscience s a bayer corporation bristol myers squibb company elan pharmaceuticals inc pharmacia corporation protein design labs inc scios inc and dow agrosciences llc and is building its internal development program in the area of oncology</text>
<text>for more information please visit the company s web site at</text>
-
 <text>www exelixis com</text>

</p>

DOM context

Link score = linear combination between page-based and context-based similarity score

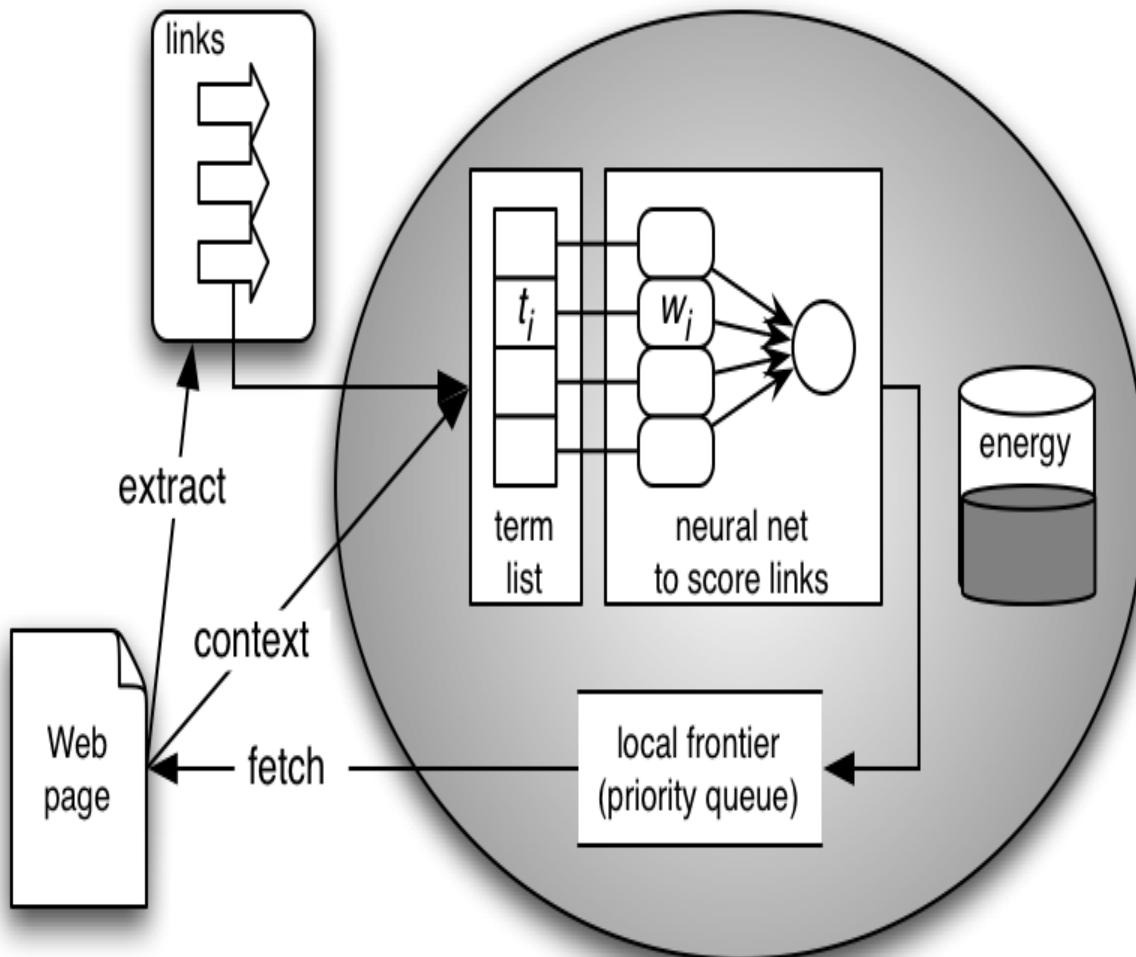


Exploration vs Exploitation

- **Best- N -First** (or BFS N)
- Rather than re-sorting the frontier every time you add links, be lazy and sort only every N pages visited
- Empirically, *being less greedy* helps crawler performance significantly: escape “local topical traps” by exploring more

InfoSpiders

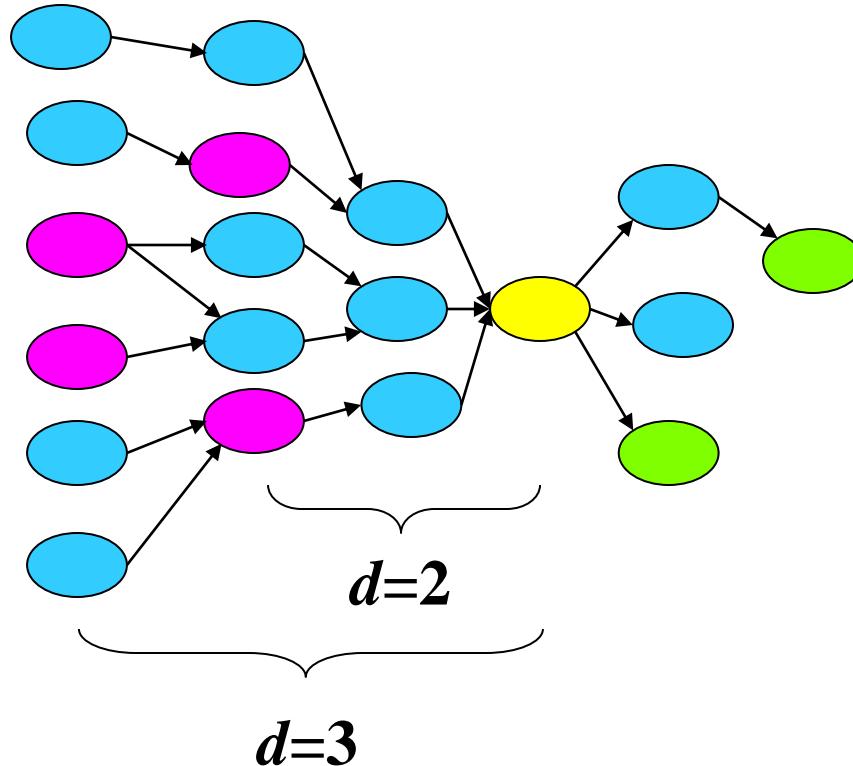
- A series of intelligent multi-agent topical crawling algorithms employing various adaptive techniques:
 - Evolutionary bias of exploration/exploitation
 - Selective query expansion
 - (Connectionist) reinforcement learning



Evaluation of topical crawlers

- Goal: build “better” crawlers to support applications
- Build an unbiased evaluation framework
 - Define common tasks of measurable difficulty
 - Identify topics, relevant targets
 - Identify appropriate performance measures
 - Effectiveness: quality of crawler pages, order, etc.
 - Efficiency: separate CPU & memory of crawler algorithms from bandwidth & common utilities

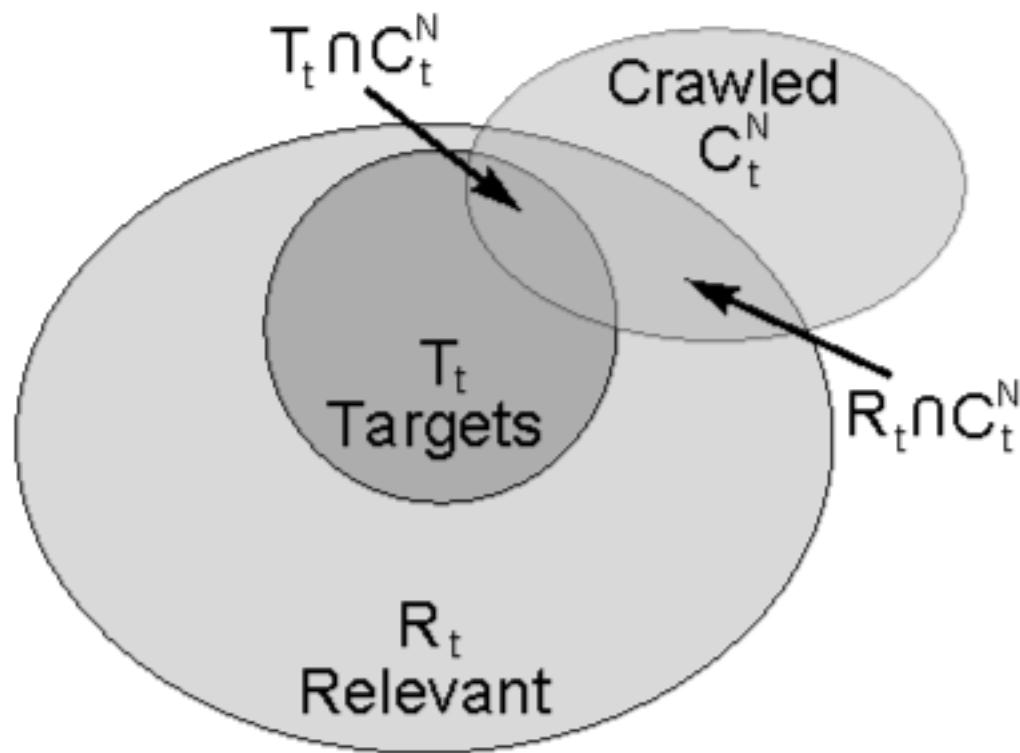
Tasks



Start from
seeds, find
targets
and/or pages
similar to
target
descriptions

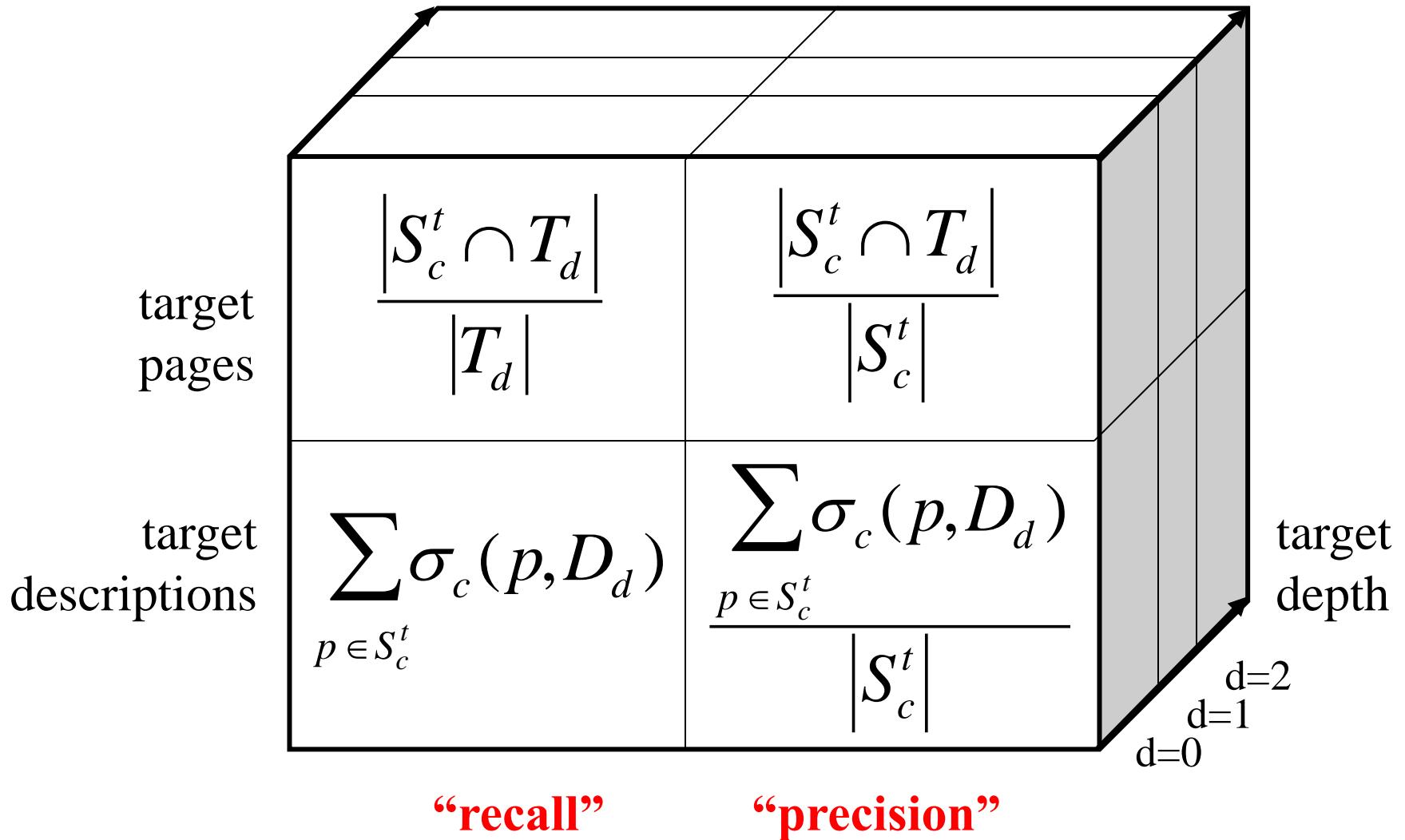
Back-crawl from targets to get seeds

Target based performance measures



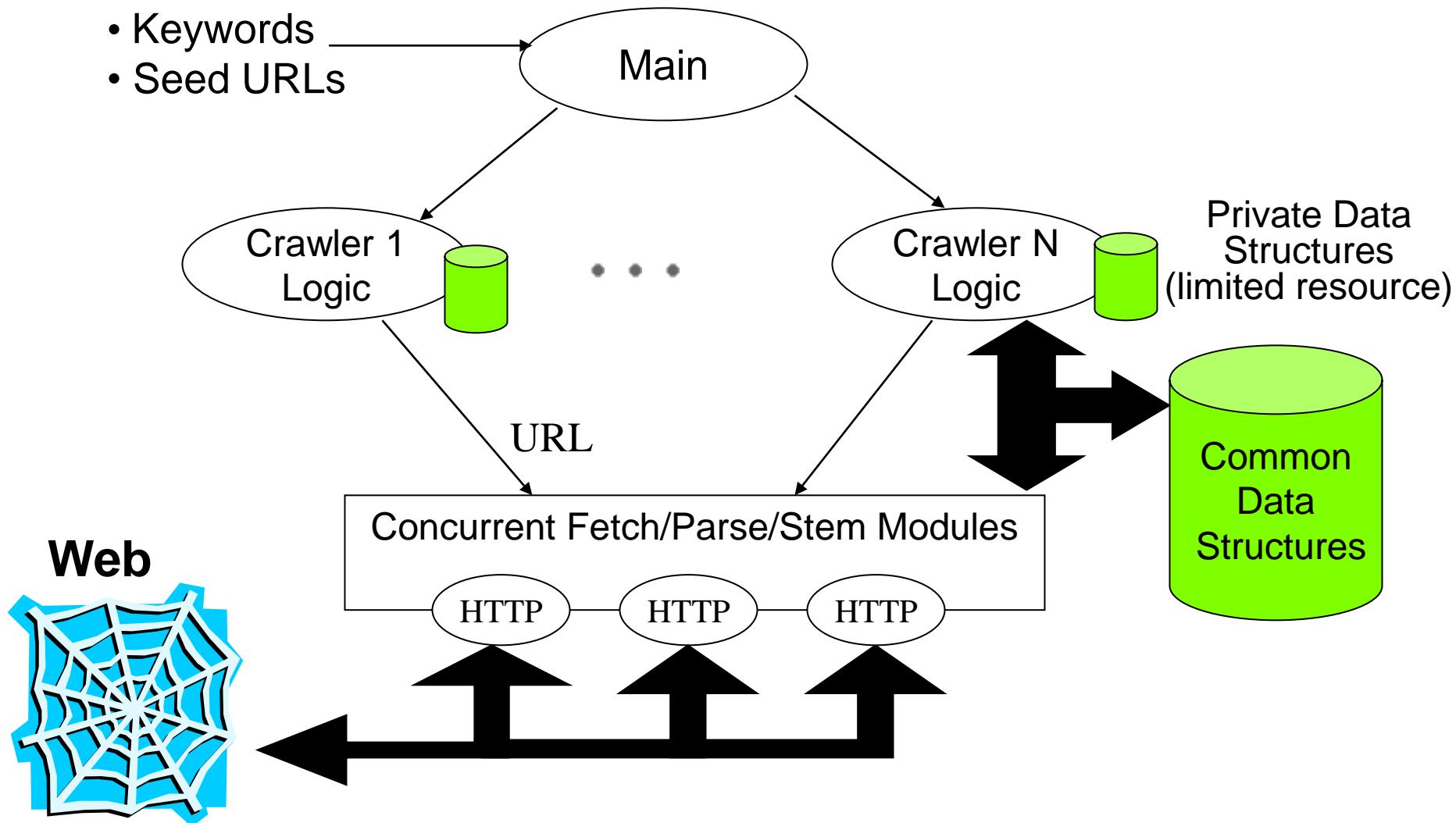
Q: What assumption are we making? A: Independence!...

Performance matrix



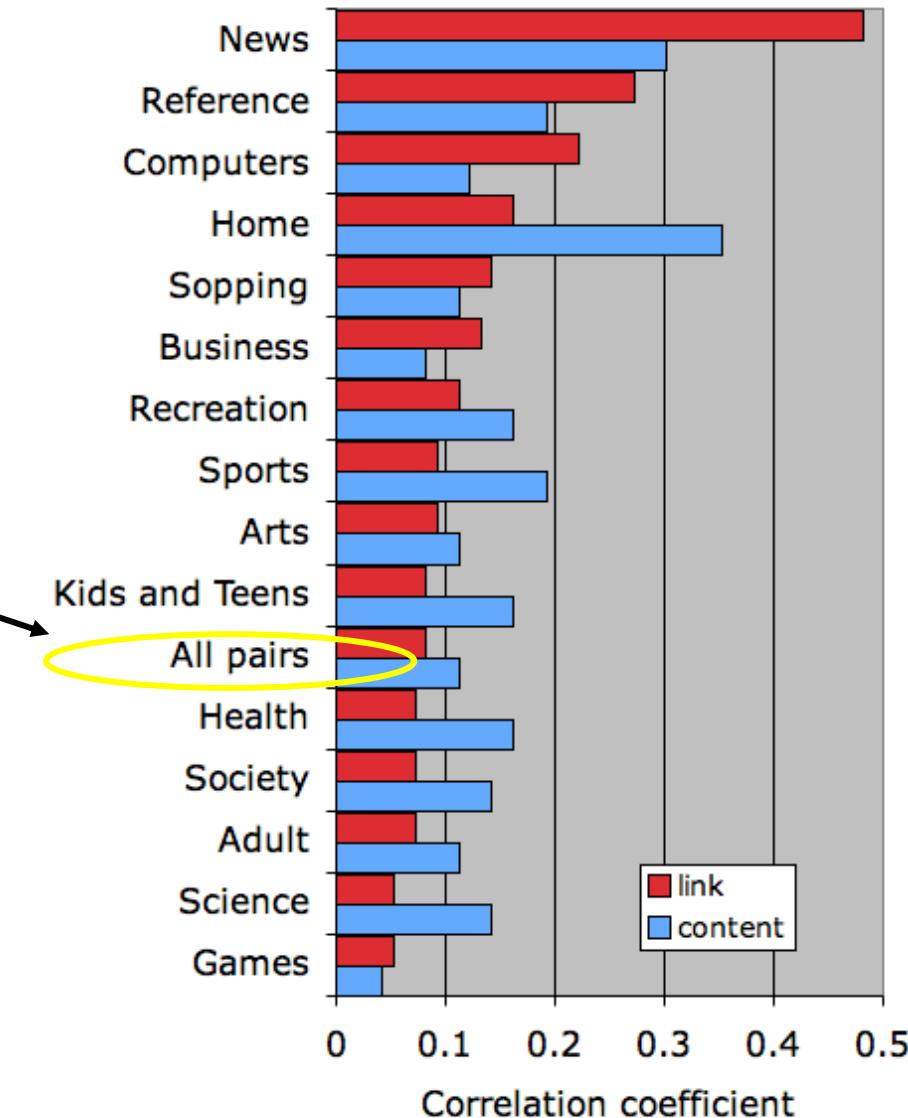
Crawling evaluation framework

- Keywords
- Seed URLs



Correlations between different similarity measures

- Semantic similarity measured from ODP, correlated with:
 - Content similarity: **TF or TF-IDF vector cosine**
 - Link similarity: **Jaccard coefficient** of (in+out) link neighborhoods
- Correlation overall is **significant but weak**
- Much stronger topical locality NEEDED



WEB MINING

Crawlers Evaluation –Inverted Indexing

TF - IDF

- **TF-IDF**, short for **Term Frequency–Inverse Document Frequency**, is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus.
- Variations of the **TF–IDF weighting scheme** are often used by search engines as ***a central tool in scoring and ranking a document's relevance*** given as user query.
- TF-IDF is one of the most popular term-weighting schemes.
- For instance, 83% of text-based recommender systems in the domain of digital libraries use **TF-IDF**.

Document 1

Term	Term Count
this	1
is	1
a	2
sample	1

Document 2

Term	Term Count
this	1
is	1
another	2
example	3

$$TT = 5$$

~~$$TF = \frac{1}{5}$$~~

$$\begin{aligned} TF(\text{this}) &= \frac{1}{5} \\ &= 0.2 \end{aligned}$$

$$\begin{aligned} TF(\text{this}) &= \frac{1}{5} \\ &= 0.2 \end{aligned}$$

$$\begin{aligned} IDF(\text{this}) &= \log_2\left(\frac{2}{2}\right) \\ &= 0 \end{aligned}$$

$$\begin{aligned} TF(\text{example}) &= \frac{3}{5} \\ &= 0.6 \end{aligned}$$

$$tf = \frac{f_N}{T}$$

"you were born with potential" and "you were born with wings"

$$tf \cdot idf = \frac{f_N}{T} \cdot \log_2\left(\frac{N}{f_N}\right)$$

~~$$TF \cdot IDF = 0.2 \cdot 0$$~~

~~$$= 0.14$$~~

$$\begin{aligned} TF \cdot IDF &= 0.2 \cdot 0 \\ &= 0 \end{aligned}$$

$$TF \cdot IDF(\text{example}) = 0.6 \cdot 0$$

$$= 0.43 \rightarrow 1$$

$$\begin{aligned} 0.43 \times 0.30 &= 0.12 \end{aligned}$$

“you were born with potential” and “you were born with wings”

TF - IDF

- Suppose we have a set of English text documents and wish to determine which document is most relevant to the query "**the brown cow**".
- A simple way to start out is by **eliminating documents** that do not contain all three words "the", "brown", and "cow", but this still leaves many documents.
- To further distinguish them, we might count the number of times each term occurs in each document; *the number of times a term occurs in a document is called its term frequency.*

TF - IDF

- Because the term "the" is so common, term frequency will tend to incorrectly emphasize documents which happen to use the word "**the**" more frequently, without giving enough weight to the more meaningful terms "**brown**" and "**cow**".
- The term "the" is not a good keyword to distinguish **relevant and non-relevant documents** and terms, unlike the less common words "brown" and "cow".
- Hence an ***inverse document frequency*** factor is incorporated which diminishes the weight of terms that occur very frequently in the document set and increases the weight of terms that occur rarely.

The TFIDF Encoding

(Term Frequency x Inverse Document Frequency)

A *term* is a word, or some other frequently occurring item

Given some term i , and a document j , the *term count* is the number of times that term i occurs in document j

Given a collection of k terms and a set D of documents, the *term n_{ij} frequency*, is:

$$tf_{ij} = \frac{n_{ij}}{\sum_{k=1}^T n_{kj}}$$

considering only the terms of interest, this is the proportion of document j that is made up from term i .

TF - IDF

Term frequency tf_{ij} is a measure of the importance of term i in document j

Inverse document frequency (which we see next) is a measure of the *general* importance of the term.

I.e. **High term frequency** for “apple” means that apple is an important word in a specific document.

But **high document frequency** (low inverse document frequency) for “apple”, given a particular set of documents, means that apple is not all that important overall, since it is in all of the documents.

TF - IDF

Inverse document frequency of term i is:

$$idf_i = \log \frac{|D|}{\{d_j : d_j \in D\}}$$

Log of: ... the number of documents in the master collection,
divided by the number of those documents that contain the term.

TFIDF encoding of a document

So, given:

- a background collection of documents
 - (e.g. 100,000 random web pages, —
all the articles we can find about cancer)
 - 100 student essays submitted as coursework
- a specific ordered list (possibly large) of terms

We can encode any document as a vector of TFIDF numbers, where the i th entry in the vector for document j is:

$$tf_{ij} \times \underbrace{idf_i}_{\checkmark}$$

Turning a document into a vector

Suppose our Master List is:

(banana, cat, dog, fish, read)

Suppose document 1 contains only:

“Bananas are grown in hot countries, and cats like bananas.”

And suppose the *background frequencies* of these words in a large random collection of documents is $(0.2, 0.1, 0.05, 0.05, 0.2)$

The document 1 vector entry for word w is:

$$\text{freqindoc}(w) \log_2(1 / \text{freq_in_bg}(w))$$

This is just a rephrasing of TFIDF, where: $\text{freqindoc}(w)$ is the frequency of w in document 1, and $\text{freq_in_bg}(w)$ is the ‘background’ frequency in our reference set of documents

Turning a document into a vector

Master list: (banana, cat, dog, fish, read)

Background frequencies: (0.2, 0.1, 0.05, 0.05, 0.2)

Document 1:

“Bananas are grown in hot countries, and cats like bananas.”

Frequencies are *proportions*. The background frequency of banana is 0.2, meaning that 20% of documents in general contain ‘banana’, or bananas, etc. (note that read includes reads, reading, reader, etc...)

The frequency of banana in document 1 is also 0.2 – why?

The TFIDF encoding of this document is:

0.464, 0.332, 0, 0, 0

Suppose another document has exactly the same vector – will it be the same document?

TF-IDF

Doc 1: Ben studies about Computers in Computer Lab

Doc 2: Steve teacher at VIT University.

Doc 3: Data scientists work on large datasets.

Query: Data Scientists

f_{Di} → Total no. of terms in document \rightarrow t_{Di}

DOC 1 Ben studies Computer Lab

tf	1	1	2	1
Ntf	0.143	0.143	0.286	0.143

DOC NO	t_{Di}
1	7
2	5
3	6

DOC 2 Steve teacher VIT University

tf	1	1	1	1
Ntf	0.2	0.2	0.2	0.2

$Ntf \rightarrow$ Normalized term frequency

DOC 3 Data Scientists work large datasets

tf	1	1	1	1	1
Ntf	0.167	0.167	0.167	0.167	0.167

$$Ntf = \frac{tf}{t_{Di}}$$

TF-IDF

$$idf = \log_2 \frac{N}{df(t)} \rightarrow$$

Total no. of documents
No. of documents having term t.

$$idf(\text{Ben}) = \log_2 \frac{3}{1} = 1.5849$$

$$idf(\text{dataset}) = \log_2 \frac{3}{1} = 1.5849$$

$$tf-idf(t,d) = tf(t,d) * idf(t,d)$$

	DOC1	DOC2	DOC3	
Data	0	0	$0.167 \times 1.5849 = 0.2646$	
Scientists	0	0	$0.167 \times 1.5849 = 0.2646$	

Jaccard's Coefficient of Link Neighbourhood

- The Jaccard Coefficient, also known as **Jaccard index** or **Jaccard similarity coefficient**, is a statistic measure used for comparing similarity of sample sets.
- It is usually denoted as $J(x, y)$ where x and y represent two different nodes in a network.
- In link prediction, all the neighbours of a node are treated as a set and the prediction is done by computing and ranking the similarity of the neighbour set of each node pair.
- This method is based on Common Neighbours method and its complexity is also $O(Nk^2)$.

Jaccard's Coefficient of Link Neighbourhood

- The mathematical expression of this method is as follows

- Score $(x, y) =$

$$\boxed{\frac{|\Gamma(x) \cap \Gamma(y)|}{|\Gamma(x) \cup \Gamma(y)|}}.$$

Jaccard's Coefficient of Link Neighbourhood

- Jaccard coefficient is one of the metrics used to compare the **similarity and diversity of sample sets**.
- It uses the ratio of the intersecting set to the union set as the measure of similarity.
- Thus it **equals to zero** if there are no intersecting elements and **equals to one** if all elements intersect.
Equation for Jaccard coefficient is

$$T = \frac{N_c}{N_a + N_b - N_c}$$

- where
- Na - number of elements in set A,
- Nb - number of elements in set B
- Nc - number of elements in intersecting set

Evaluation of Crawlers

- Goal: build “better” crawlers to support applications
- Build an unbiased evaluation framework
- Define common tasks of measurable difficulty
- Identify topics, relevant targets
- Identify appropriate performance measures
- **Effectiveness:** quality of crawler pages, order, etc.
- **Efficiency:** separate CPU & memory of crawler algorithms from bandwidth & common utilities
- *Perhaps the most crucial evaluation of a crawler is to measure the rate at which relevant web pages are acquired and how effectively irrelevant web pages are filtered out from the crawler.*

Evaluation of Crawlers

- With this knowledge, we could estimate the **precision and recall** of a crawler after crawling n web pages.
- The precision would be the fraction of pages crawled that are relevant to the topic and recall would be the fraction of relevant pages crawled.
- However, the relevant set for any given topic is unknown in the web, so the true recall is hard to measure.
- Therefore, we adopt **harvest rate and target recall** to evaluate the performance of the crawler.
- In case we have boolean relevance scores, we could measure the rate at which “good” pages are found; if 100 relevant pages are found in the first 500 pages crawled, we have an acquisition rate or **harvest rate** of 20% at 500 pages.

Evaluation of Crawlers

- The harvest rate is the fraction of web pages crawled that are relevant to the given topic, which measures how well it is doing at rejecting irrelevant web pages. The expression is given by:

$$\text{Harvest rate} = \frac{\sum_{i \in V} r_i}{|V|},$$

- where
- V is the number of web pages crawled by a crawler in current;
- r_i is the relevance between web page i and the given topic, and the value of r_i can only be 0 or 1.
- If relevant, then $r_i = 1$; otherwise $r_i = 0$.

Evaluation of Topical Crawlers

- The **target recall** is the fraction of relevant pages crawled, which measures how well it is doing at finding all the **relevant web pages**.
- However, the relevant set for any given topic is unknown in the Web, so the **true target recall is hard to measure**.
- In view of this situation, we delineate a specific network, where given a set of seed URLs and a certain depth, the range reached by a **crawler using breadth-first crawling strategy is the virtual Web**.
- We assume that the target set T is the relevant set in the virtual Web; C_t is the set of first t pages crawled. The expression is given by:

$$\text{Target recall} = \frac{|T \cap C_t|}{|T|}.$$

Evaluation of Topical Crawlers

Another measure from information retrieval that has been applied to crawler evaluation is **search length**,

- defined as the number of pages (or the number of irrelevant pages) crawled before a certain percentage of the relevant pages are found.
- Search length is a kind of the reciprocal of precision for a preset level of recall.

Evaluation of Topical Crawlers

- A second approach is to split the set of known relevant pages into two sets; **one set can be used as seeds, the other as targets.**
- While there is no guarantee that the **targets** are reachable from the **seeds**, this approach is significantly simpler because **no back-crawl** is necessary.
- Another advantage is that each of the two relevant subsets can be used in turn as **seeds and targets.**
- In this way, one can measure the overlap between the pages crawled starting from the two disjoint sets.

Crawler Etiquette

- **Identify yourself**
 - Use ‘User-Agent’ HTTP header to identify crawler, website with description of crawler and contact information for crawler developer
 - ❑ Use ‘From’ HTTP header to specify crawler developer email
 - ❑ Do not disguise crawler as a browser by using their ‘User-Agent’ string
- Always check that HTTP requests are successful, and in case of error, use HTTP error code to determine and immediately address problem
- Pay attention to anything that may lead to too many requests to any one server, even unwillingly, e.g.:
 - ✓ redirection loops
 - ✓ spider traps

Crawler Ettiquette : Server Aspects

- ***Spread the load, do not overwhelm a server***
- Make sure that no more than some max. number of requests to any single server per unit time, say < 1/second
- ***Honor the Robot Exclusion Protocol***
- A server can specify which parts of its document tree any crawler is or is not allowed to crawl by a file named ‘robots.txt’ placed in the HTTP root directory, **e.g.** <http://www.indiana.edu/robots.txt>
- Crawler should always check, parse, and obey this file before sending any requests to a server

Crawler Ethics Issues

- Is compliance with robot exclusion a matter of law?
- No! Compliance is voluntary, but if you do not comply, you may be blocked
- Someone (unsuccessfully) , Internet Archive over a robots.txt related issue
- Some crawlers disguise themselves
- Using false User-Agent
- Randomizing access frequency to look like a human/browser
- **Example: click fraud for ads**

Crawler Ethics Issues

- Servers can disguise themselves, too
- **Cloaking:** present different content based on UserAgent
 - E.g. stuff keywords on version of page shown to search engine crawler
- Search engines do not look kindly on this type of “spamdexing” and remove from their index sites that perform such abuse

Module - 3

Static and Dynamic Inverted Index

What is Inverted Indices?

- When the crawl is complete, the search engine builds, for **each and every** word, an **inverted index**.
- An inverted index is a list of all documents **containing** that word
 - The index may be a **bit vector**
 - It may also contain the location(s) of the word in the document
- Word: any word in any language, plus misspelling, plus any sequence of characters surrounded by punctuation!
⇒Hundreds of millions of words

Why Inverted Index?

- The basic method of Web search and traditional IR is to find documents that contain the **terms in the user query**.
- Given a user query, one option is to scan the document database sequentially to find the documents that contain the query terms.
- However, this method is obviously impractical for a large collection, such as the Web.
- Another option is to build some **data structures** (called **indices**) from the document collection
- There are many index schemes for this text

Why Inverted Index?

- Efficient **data structures** needed to process large document collections quickly
- How do we store documents in order to maximize **retrieval performance**?
 - We must avoid linear scans of text at query time
 - We must index documents in advance

Dynamic Indexing

- Optimization of an internal file at each time for **retrieval queries**.
- This means that **new terms need to be added to the dictionary**, and postings lists need to be updated for existing terms.
- The simplest way to achieve this is to periodically **reconstruct the index from scratch**.
- This is a good solution if the number of changes over time is small and a ***delay in making new documents searchable is acceptable*** - and if enough resources are available to construct a new index while the old one is still available for querying.

Dynamic Indexing

- If there is a requirement that new documents be included quickly, one solution is to maintain two indexes:
 - ❖ a **large main index**
 - ❖ a **small auxiliary index** that stores new documents.
- The auxiliary index is kept in memory. Searches are run across both indexes and results merged.
- Deletions are stored in an invalidation **bit vector**. We can then filter out deleted documents before returning the search result.
- Documents are **updated by deleting and reinserting them**.
- Each time the **auxiliary index becomes too large**, we **merge it into the main index**.
- The **cost** of this merging operation depends on how we store the index in the file system.

Dynamic Indexing : Need for Compression?

- In this scheme, the reason for keeping the auxiliary index is to reduce the number of disk seeks required over time.
- Unfortunately, the one-file-per-postings-list scheme is infeasible
- A natural solution to this is **index compression**, which aims to represent the same information with fewer bits or bytes

WEB MINING

Module - 3

**Index Compression – Unary, Gamma, Delta &
Golomb Coding**

Need for Compression?

- An inverted index can be **very large**.
- In order to speed up the search, it should reside in memory as much as possible to **avoid disk I/O**.
- Done by reducing the index size and it is an important issue.
- A natural solution to this is **index compression**, which aims to represent the same information with fewer bits or bytes.
- Using compression, the size of an inverted index can be reduced dramatically.
- In the **lossless compression**, the original index can also be reconstructed exactly using the compressed version.
- Since all the information is represented with positive integers, **integer compression** techniques is the main focus.

Compression Types

- There are generally two classes of compression schemes for inverted lists: the **variable-bit scheme** and the **variable-byte scheme**.
- In the variable-bit (also called **bitwise**) scheme, an integer is represented with an **integral number of bits**.
- *Bitwise methods include*
 - **unary coding**
 - **Elias gamma coding**
 - **delta coding**
 - **Golomb coding.**

Compression Types

- In the variable-byte scheme, an integer is stored in an integral number of bytes, where each byte has 8 bits.
- A simple bytewise scheme is the **variable-byte coding**.
- These coding schemes basically map integers onto **self-delimiting binary codewords (bits)**, i.e., the start bit and the end bit of each integer can be detected with no additional delimiters or markers.

Unary Coding

- Unary coding is simple. It represents a number x with $x-1$ bits of zeros followed by a **bit of one**.
- For example, 5 is represented as 00001.
- The one bit is simply the **delimiter**.
- *Decoding is also straightforward.*
- This scheme is effective for very small numbers, but **wasteful for large numbers**.
- It is thus seldom used alone in practice.

Decimal	Unary
1	1
2	01
3	001
4	0001
5	00001
6	000001
7	0000001
8	00000001
9	000000001
10	0000000001

Elias Gamma Coding

Coding:-

- The coding can also be described with the following two steps:
 1. Write x in binary.
 2. Subtract 1 from the number of bits written in step 1 and prepend that many zeros.

Example:-

- The number 9 is represented by 0001001. we first write 9 in binary, which is 1001 with 4 bits, and then prepend three zeros

Elias Gamma Coding

Decimal	Elias Gamma
1	1
2	0 10
3	0 11
4	00 100
5	00 101
6	00 110
7	00 111
8	000 1000
9	000 1001
10	000 1010

Elias Gamma Coding

Decoding:-

We decode an **Elias gamma-coded** integer in two steps:

1. Read and count zeroes from the stream until we reach the first one. Call this count of zeroes K .
2. Consider the one that was reached to be the first digit of the integer, with a value of $2K$, read the remaining K bits of the integer.

Example:-

- ❖ To decompress **0001001**, we first read all zero bits from the beginning until we see a bit of 1.
- ❖ We have $K = 3$ zero bits. We then include the 1 bit with the following 3 bits, which give us 1001.

Elias Gamma Coding

- Gamma coding is efficient for small integers but is not suited to large integers for which the parameterized Golomb code or the Elias delta code is more suitable.

Elias Delta Coding

Coding:-

- In the Elias delta coding, a positive integer x is stored with the gamma code representation of $1+\lfloor \log_2 x \rfloor$, followed by the binary representation of x less the most significant bit.

Example:-

- Let us code the number 9. Since $1+\lfloor \log_2 9 \rfloor = 4$, we have its gamma code 00100 for 4. Since 9's binary representation less the most significant bit is 001, we have the delta code of 001 00001 for 9.

Elias Delta Coding

Selector	Offset
Gamma	Binary

- 1) Gamma Code representation of $1 + \lfloor \log_2 x \rfloor$
(selector)
- 2) Binary representation of 'x' less the MSB
(offset)

Example:

$$x = 9$$

① $1 + \lfloor \log_2 9 \rfloor = 1 + 3 = 4$ in Gamma code

4 in Gamma code

1) $1 + \log_2 x$ in unary

$1 + 2$ in unary
= 001

2) Write x in binary

$4 \Rightarrow \begin{matrix} 1 \\ 0 \\ 0 \end{matrix}$
remove MSB = 00

001	00
-----	----

② Binary of 9 = 1001

less MSB $\times 1001$ ← offset

00100	001
-------	-----

Elias Delta Coding

Decimal	Elias Delta
1	1
2	0 100
3	0 101
4	0 1100
5	0 1101
6	0 1110
7	0 1111
8	00 100000
9	00 100001
10	00 100010

Elias Delta Coding

- An equivalent way to express the same process:
 - Separate X into the highest power of 2 it contains (2^N) and the remaining N binary digits.
 - Encode $N+1$ with Elias gamma coding.
 - Append the remaining N binary digits to this representation of $N+1$.

Number	N	$N+1$	δ encoding
$1 = 2^0$	0	1	1
$2 = 2^1 + 0$	1	2	0 1 0 0
$3 = 2^1 + 1$	1	2	0 1 0 1
$4 = 2^2 + 0$	2	3	0 1 1 00
$5 = 2^2 + 1$	2	3	0 1 1 01
$6 = 2^2 + 2$	2	3	0 1 1 10
$7 = 2^2 + 3$	2	3	0 1 1 11
$8 = 2^3 + 0$	3	4	00 1 00 000
$9 = 2^3 + 1$	3	4	00 1 00 001
$10 = 2^3 + 2$	3	4	00 1 00 010

Elias Delta Coding

Decoding:-

Specifically, we use the following steps:

1. Read and count zeroes from the stream until you reach the first one. **Call this count of zeroes L .**
2. Considering the one that was reached to be the first bit of an integer, with a value of $2L$, read the remaining L digits of the integer. **This is the integer M .**
3. Put a one in the first place of our final output, representing the value $2M$. Read and append the following $M-1$ bits.

Elias Delta Coding

Decoding:-

Example:-

- We want to decode 00100001. We can see that $L = 2$ after step 1, and after step 2, we have read and consumed 5 bits.
- We also obtain $M = 4$ (100 in binary).
- Finally, we prepend 1 to the $M-1$ bits (which is 001) to give 1001, which is 9 in binary.

Decoding

- (ie) L
- 1) Read and count zero's until one reached
 - 2) Read ' L ' bits next from the input
 - 3) Convert it to decimal and get the number of bits (M) $(M-1)$ read from the end.
 - 4) Prepend a '1'

Example:

00100001

- 1) Read till '1' $L = 2$
- 2) Read from '1' ' L ' bits = 100
- 3) Now read $(M-1)$ bits from LSB
and prepend '1'
 $M = 4$

001 from LSB

prepend 1 $\Rightarrow 1001 = 9$

Golomb Coding

The Golomb coding is a form of **parameterized coding** in which integers to be coded are stored as **values relative to a constant b** .

Coding:-

A positive integer x is represented in two parts:

1. The first part is a unary representation of $q+1$, where q is the quotient $\lfloor (x/b) \rfloor$, and
 2. The second part is a special binary representation of the remainder $r = x - qb$. Note that there are b possible remainders.
- For example, if $b = 3$, the possible remainders will be 0, 1, and 2.

Golomb Coding

- write the first few remainders using $\lfloor \log_2 b \rfloor$ bits and the rest using $\lceil \log_2 b \rceil$ bits.
- We must do so such that the decoder knows when $\lfloor \log_2 b \rfloor$ bits are used and when $\lceil \log_2 b \rceil$ bits are used

Golomb Coding : Example

- Let $i = \lfloor \log_2 b \rfloor$. We code the first d remainders using i bits, $d = 2^i + 1 - b$.
- For $b = 3$, to code $x = 9$, we have the quotient $q = \lfloor 9/3 \rfloor = 3$.
- For remainder, we have $i = \lfloor \log_2 3 \rfloor = 1$ and $d = 1$.
- Note that for $b = 3$, there are three remainders, i.e., 0, 1, and 2, which are coded as 0, 10, and 11 respectively. T
- The remainder for 9 is $r = 9 - 3 \times 3 = 0$.
- The final code for 9 is 00010.
- We can see that the first d remainders are standard binary codes, but the rest are not. They are generated using a tree instead.

Coding for $b=3$ and $b=5$

Decimal	Unary	Elias Gamma	Elias Delta	Golomb ($b = 3$)	Golomb ($b = 10$)	Variable byte
1	1	1	1	1 10	1 001	0000001 0
2	01	0 10	0 100	1 11	1 010	0000010 0
3	001	0 11	0 101	01 0	1 011	0000011 0
4	0001	00 100	0 1100	01 10	1 100	0000100 0
5	00001	00 101	0 1101	01 11	1 101	0000101 0
6	000001	00 110	0 1110	001 0	1 1100	0000110 0
7	0000001	00 111	0 1111	001 10	1 1101	0000111 0
8	00000001	000 1000	00 100000	001 11	1 1110	0001000 0
9	000000001	000 1001	00 100001	0001 0	1 1111	0001001 0
10	0000000001	000 1010	00 100010	0001 10	01 000	0001010 0

Decoding

Decoding: To decode a Golomb-coded integer x , we use the following steps:

1. Decode unary-coded quotient q (the relevant bits are consumed).
2. Compute $i = \lfloor \log_2 b \rfloor$ and $d = 2^{i+1} - b$.
3. Retrieve the next i bits and assign it to r .
4. If $r \geq d$ then
 - retrieve one more bit and append it to r at the end;
 - $r = r - d$.
5. Return $x = qb + r$.

Some explanation is in order for step 4. As we discussed above, if $r \geq d$ we need $i+1$ bits to code the remainder. The first line of step 4 retrieves the additional bit and appends it to r . The second line obtains the true value of the remainder r .

Variable-Byte Coding

- In this method, seven bits in each byte are used to code an integer, with the least significant bit set to 0 in the last byte, or to 1 if further bytes follow.
- In this way, small integers are represented efficiently.
- For example, 135 is represented in two bytes, since it lies in the range 27 and 214, as 00000011 00001110.

Variable Byte Coding

Decimal	Unary	Elias Gamma	Elias Delta	Golomb ($b = 3$)	Golomb ($b = 10$)	Variable byte
1	1	1	1	1 10	1 001	0000001 0
2	01	0 10	0 100	1 11	1 010	0000010 0
3	001	0 11	0 101	01 0	1 011	0000011 0
4	0001	00 100	0 1100	01 10	1 100	0000100 0
5	00001	00 101	0 1101	01 11	1 101	0000101 0
6	000001	00 110	0 1110	001 0	1 1100	0000110 0
7	0000001	00 111	0 1111	001 10	1 1101	0000111 0
8	00000001	000 1000	00 100000	001 11	1 1110	0001000 0
9	000000001	000 1001	00 100001	0001 0	1 1111	0001001 0
10	0000000001	000 1010	00 100010	0001 10	01 000	0001010 0

Variable Length Decoding

Decoding is performed in two steps:

1. Read all bytes until a byte with the zero last bit is seen.
2. Remove the least significant bit from each byte read so far and concatenate the remaining bits.

For example,
is decoded to 0000001**1** 0000111**0**
 0000001 0000111

Latent Semantic Indexing

- The retrieval models discussed so far are based on keyword or term matching, i.e., matching terms in the user query with those in the documents.
- However, many concepts or objects can be described in multiple ways (using different words) due to the context and people's language habits.
- If a user query uses different words from the words used in a document, the document will not be retrieved although it may be relevant because the document uses some synonym's of the words in the user query.
- This causes low recall. For example, “picture”, “image” and “photo” are **synonyms** in the context of digital cameras.
- If the user query only has the word “picture”, relevant documents that contain “image” or “photo” but not “picture” will not be retrieved.

Latent Semantic Indexing

- Latent semantic indexing (LSI), proposed by Deerwester aims to deal with this problem through the identification of statistical associations of terms.
- It then uses a statistical technique, called **singular value decomposition** (SVD), to estimate this latent structure, and to remove the “noise”.
- The results of this decomposition are descriptions of terms and documents based on the latent semantic structure derived from SVD.
- This structure is also called the **hidden “concept” space**, which associates syntactically different but semantically similar terms and documents.

Latent Semantic Indexing

- Let D be the text collection, the number of distinctive words in D be m and the number of documents in D be n .
- LSI starts with an mxn term document matrix \mathbf{A} .
- Each row of \mathbf{A} represents a term and each column represents a document.
- The matrix may be computed in various ways, e.g.,
- using term frequency or TF-IDF values.
- We use term frequency as an example in this section.
- Thus, each entry or cell of the matrix \mathbf{A} , denoted by A_{ij} , is the number of times that term i occurs in document j .

Singular Value Decomposition

- What SVD does is to factor matrix A (a $m \times n$ matrix) into the product of three matrices, i.e.,

$$A = U \Sigma V^T$$

where

U is a $m \times r$ matrix and its columns, called **left singular vectors**, are eigenvectors associated with the r non-zero eigenvalues of AA^T . Furthermore, the columns of U are unit orthogonal vectors, i.e., $U^T U = I$ (identity matrix).

V is an $n \times r$ matrix and its columns, called **right singular vectors**, are eigenvectors associated with the r non-zero eigenvalues of $A^T A$. The columns of V are also unit orthogonal vectors, i.e., $V^T V = I$.

Σ is a $r \times r$ diagonal matrix, $\Sigma = \text{diag}(\sigma_1, \sigma_2, \dots, \sigma_r)$, $\sigma_i > 0$. $\sigma_1, \sigma_2, \dots$, and σ_r , called **singular values**, are the non-negative square roots of the r (non-zero) eigenvalues of AA^T . They are arranged in decreasing order, i.e., $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_r > 0$.

TF-IDF & Jaccard similarity

TF - IDF

- **TF-IDF**, short for **Term Frequency–Inverse Document Frequency**, is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus.
- Variations of the **TF-IDF weighting scheme** are often used by search engines as *a central tool in scoring and ranking a document's relevance* given as user query.
- TF-IDF is one of the most popular term-weighting schemes.
- For instance, 83% of text-based recommender systems in the domain of digital libraries use **TF-IDF**.

Document 1

Term	Term Count
this	1
is	1
a	2
sample	1

Document 2

Term	Term Count
this	1
is	1
another	2
example	3

$$tf = \frac{t_n}{T}$$

"you were born with potential" and "you were born with wings"



$$tf \cdot idf = \frac{t}{T} \cdot \log \left(\frac{T}{t} \right)$$

$$T = 5$$

$$T = 7$$

$$tf(\text{this}) = \frac{1}{5} = 0.2$$

$$tf(\text{this}) = \frac{1}{7} = 0.14$$

$$idf(\text{this}) = \log \left(\frac{2}{2} \right) = 0$$

$$tf(\text{is}) = 0$$

$$tf-idf(\text{this}) = 0 - 0.14$$

$$tf-idf(\text{another}) = 0 - 0.14$$

$$tf(\text{example}) = \frac{1}{3} = 0.33$$

$$0.43 \times 0.33 = 0.12$$

TF - IDF

- Suppose we have a set of English text documents and wish to determine which document is most relevant to the query "**the brown cow**".
- A simple way to start out is by **eliminating documents** that do not contain all three words "**the**", "**brown**", and "**cow**", but this still leaves many documents.
- To further distinguish them, we might count the number of times each term occurs in each document; *the number of times a term occurs in a document is called its term frequency.*

TF - IDF

- Because the term "the" is so common, term frequency will tend to incorrectly emphasize documents which happen to use the word "**the**" more frequently, without giving enough weight to the more meaningful terms "**brown**" and "**cow**".
- The term "the" is not a good keyword to distinguish **relevant and non-relevant documents** and terms, unlike the less common words "brown" and "cow".
- Hence an ***inverse document frequency*** factor is incorporated which diminishes the weight of terms that occur very frequently in the document set and increases the weight of terms that occur rarely.

The TFIDF Encoding

(Term Frequency x Inverse Document Frequency)

A *term* is a word, or some other frequently occurring item

Given some term i , and a document j , the *term count* is the number of times that term i occurs in document j

Given a collection of k terms and a set D of documents, the *term frequency*, is:

$$tf_{ij} = \frac{n_{ij}}{\sum_{k=1}^T n_{kj}}$$

considering only the terms of interest, this is the proportion of document j that is made up from term i .

TF - IDF

Term frequency

tf_{ij} is a measure of the importance of term i in

document j

Inverse document frequency (which we see next) is a measure of the *general* importance of the term.

I.e. **High term frequency** for “apple” means that apple is an important word in a specific document.

But **high document frequency** (low inverse document frequency) for “apple”, given a particular set of documents, means that apple is not all that important overall, since it is in all of the documents.

TF - IDF

Inverse document frequency of term i is:

$$idf_i = \log \frac{|D|}{\{d_j : d_j \in D\}}$$

Log of: ... the number of documents in the master collection,
divided by the number of those documents that contain the term.

TFIDF encoding of a document

So, given:

- a background collection of documents
 - (e.g. 100,000 random web pages, —
 - all the articles we can find about cancer
 - 100 student essays submitted as coursework
- a specific ordered list (possibly large) of terms

We can encode any document as a vector of TFIDF numbers, where the i th entry in the vector for document j is:

$$tf_{ij} \times \underbrace{idf_i}_{\checkmark}$$

Turning a document into a vector

Suppose our Master List is:

(banana, cat, dog, fish, read)

Suppose document 1 contains only:

“Bananas are grown in hot countries, and cats like bananas.”

And suppose the *background frequencies* of these words in a large random collection of documents is (0.2, 0.1, 0.05, 0.05, 0.2)

The document 1 vector entry for word w is:

$$\text{freqindoc}(w) \log_2(1 / \text{freq_in_bg}(w))$$

This is just a rephrasing of TFIDF, where: freqindoc(w) is the frequency of w in document 1, and freq_in_bg(w) is the ‘background’ frequency in our reference set of documents

Turning a document into a vector

Master list: (banana, cat, dog, fish, read)

Background frequencies: (0.2, 0.1, 0.05, 0.05, 0.2)

Document 1:

“Bananas are grown in hot countries, and cats like bananas.”

Frequencies are *proportions*. The background frequency of banana is 0.2, meaning that 20% of documents in general contain ‘banana’, or bananas, etc. (note that read includes reads, reading, reader, etc...)

The frequency of banana in document 1 is also 0.2 – why?

The TFIDF encoding of this document is:

0.464, 0.332, 0, 0, 0

Suppose another document has exactly the same vector – will it be the same document?

TF-IDF

Doc 1: Ben studies about Computers in Computer Lab

Doc 2: Steve teacher at VIT University.

Doc 3: Data scientists work on Large datasets.

Query: Data Scientists

Doc 1 Ben studies Computer Lab

	tf	1	1	2	1
Ntf	0.143	0.143	0.286	0.143	

Doc 2 Steve teacher VIT University

	tf	1	1	1	1
Ntf	0.2	0.2	0.2	0.2	

Doc 3 Data Scientists work large datasets

	tf	1	1	1	1	1
Ntf	0.167	0.167	0.167	0.167	0.167	

t_{Di} → Total no. of terms in documents $> Di$

DOC NO	t_{Di}
1	7
2	5
3	6

$Ntf \rightarrow$ Normalized term frequency

$$Ntf = \frac{tf}{t_{Di}}$$

TF-IDF

$$idf = \log \frac{N}{df(t)} \rightarrow \begin{array}{l} N \rightarrow \text{Total no. of documents} \\ df(t) \rightarrow \text{No. of documents having term } t. \end{array}$$

$$idf(\text{Ben}) = \log_2 \frac{3}{1} = 1.5849$$

$$idf(\text{dataset}) = \log_2 \frac{3}{1} = 1.5849$$

$$tf \cdot idf(t,d) = tf(t,d) * idf(t,d)$$

	DOC1	DOC2	DOC3
Data	0	0	$0.167 \times 1.5849 = 0.2646$
Scientists	0	0	$0.167 \times 1.5849 = 0.2646$

Jaccard's Coefficient of Link Neighbourhood

- The Jaccard Coefficient, also known as **Jaccard index or Jaccard similarity coefficient**, is a statistic measure used for comparing similarity of sample sets.
- It is usually denoted as $J(x, y)$ where x and y represent two different nodes in a network.
- In link prediction, all the neighbours of a node are treated as a set and the prediction is done by computing and ranking the similarity of the neighbour set of each node pair.
- This method is based on Common Neighbours method and its complexity is also $O(Nk^2)$.

Jaccard's Coefficient of Link Neighbourhood

- The mathematical expression of this method is as follows
- Score $(x, y) =$

$$\left| \frac{\Gamma(x) \cap \Gamma(y)}{\Gamma(x) \cup \Gamma(y)} \right|.$$

Jaccard's Coefficient of Link Neighbourhood

- Jaccard coefficient is one of the metrics used to compare the **similarity and diversity of sample sets**.
- It uses the ratio of the intersecting set to the union set as the measure of similarity.
- Thus it **equals to zero** if there are no intersecting elements and **equals to one** if all elements intersect. Equation for Jaccard coefficient is

- where
 - Na - number of elements
 - Nb - number of elements
 - Nc - number of elements in intersecting set
- $$T = \frac{N_c}{N_a + N_b - N_c}$$

Evaluation of Crawlers

- Goal: build “better” crawlers to support applications
- Build an unbiased evaluation framework
- Define common tasks of measurable difficulty
- Identify topics, relevant targets
- Identify appropriate performance measures
- **Effectiveness:** quality of crawler pages, order, etc.
- **Efficiency:** separate CPU & memory of crawler algorithms from bandwidth & common utilities
- *Perhaps the most crucial evaluation of a crawler is to measure the rate at which relevant web pages are acquired and how effectively irrelevant web pages are filtered out from the crawler.*

Evaluation of Crawlers

- With this knowledge, we could estimate the **precision and recall** of a crawler after crawling n web pages.
- The precision would be the fraction of pages crawled that are relevant to the topic and recall would be the fraction of relevant pages crawled.
- However, the relevant set for any given topic is unknown in the web, so the true recall is hard to measure.
- Therefore, we adopt **harvest rate and target recall** to evaluate the performance of the crawler.
- In case we have boolean relevance scores, we could measure the rate at which “good” pages are found; if 100 relevant pages are found in the first 500 pages crawled, we have an acquisition rate or **harvest rate** of 20% at 500 pages.

Evaluation of Crawlers

- The harvest rate is the fraction of web pages crawled that are relevant to the given topic, which measures how well it is doing at rejecting irrelevant web pages. The expression is given by:

$$\text{Harvest rate} = \frac{\sum_{i \in V} r_i}{|V|},$$

- where
- V is the number of web pages crawled by a crawler in current;
- r_i is the relevance between web page i and the given topic, and the value of r_i can only be 0 or 1.
- If relevant, then $r_i = 1$; otherwise $r_i = 0$.

Evaluation of Topical Crawlers

- The **target recall** is the fraction of relevant pages crawled, which measures how well it is doing at finding all the **relevant web pages**.
- However, the relevant set for any given topic is unknown in the Web, so the **true target recall is hard to measure**.
- In view of this situation, we delineate a specific network, where given a set of seed URLs and a certain depth, the range reached by a **crawler using breadth-first crawling strategy is the virtual Web**.
- We assume that the target set T is the relevant set in the virtual Web; C_t is the set of first t pages crawled. The expression is given by:

$$\text{Target recall} = \frac{|T \cap C_t|}{|T|}.$$

Evaluation of Topical Crawlers

Another measure from information retrieval that has been applied to crawler evaluation is **search length**,

- defined as the number of pages (or the number of irrelevant pages) crawled before a certain percentage of the relevant pages are found.
- Search length is a kind of the reciprocal of precision for a preset level of recall.

Evaluation of Topical Crawlers

- A second approach is to split the set of known relevant pages into two sets; one set can be used as seeds, the other as targets.
- While there is no guarantee that the targets are reachable from the seeds, this approach is significantly simpler because no back-crawl is necessary.
- Another advantage is that each of the two relevant subsets can be used in turn as seeds and targets.
- In this way, one can measure the overlap between the pages crawled starting from the two disjoint sets.

Crawler Etiquette

- **Identify yourself**
 - Use ‘User-Agent’ HTTP header to identify crawler, website with description of crawler and contact information for crawler developer
 - ❑ Use ‘From’ HTTP header to specify crawler developer email
 - ❑ Do not disguise crawler as a browser by using their ‘User-Agent’ string
- Always check that HTTP requests are successful, and in case of error, use HTTP error code to determine and immediately address problem
- Pay attention to anything that may lead to too many requests to any one server, even unwillingly, e.g.:
 - ✓ redirection loops
 - ✓ spider traps

Crawler Ettiquette : Server Aspects

- *Spread the load, do not overwhelm a server*
- Make sure that no more than some max. number of requests to any single server per unit time, say < 1/second
- ***Honor the Robot Exclusion Protocol***
- A server can specify which parts of its document tree any crawler is or is not allowed to crawl by a file named ‘robots.txt’ placed in the HTTP root directory, e.g.
<http://www.indiana.edu/robots.txt>
- Crawler should always check, parse, and obey this file before sending any requests to a server

Crawler Ethics Issues

- Is compliance with robot exclusion a matter of law?
- No! Compliance is voluntary, but if you do not comply, you may be blocked
- Someone (unsuccessfully) , Internet Archive over a robots.txt related issue
- Some crawlers disguise themselves
- Using false User-Agent
- Randomizing access frequency to look like a human/browser
- **Example: click fraud for ads**

Crawler Ethics Issues

- Servers can disguise themselves, too
- **Cloaking:** present different content based on UserAgent
 - E.g. stuff keywords on version of page shown to search engine crawler
- Search engines do not look kindly on this type of “spamdexing” and remove from their index sites that perform such abuse

TF-IDF

Term frequency-Inverted Document Frequency

Term frequency-tf

- The term frequency $\text{tf}_{t,d}$ of **term t** in **document d** is defined as the number of times that **t occurs in d** .
- We want to use tf when computing query-document match scores. But how?
- Raw term frequency is not what we want:
 - A document with 10 occurrences of the term is more relevant than a document with 1 occurrence of the term.
 - But not 10 times more relevant.
- Relevance does not increase proportionally with term frequency.

Document frequency

- Rare terms are more informative than frequent terms
 - Recall stop words
- Consider a term in the query that is rare in the collection (e.g., *arachnocentric*)
- A document containing this term is very likely to be relevant to the query *arachnocentric*
 - We want a high weight for rare terms like *arachnocentric*.

Document frequency, continued

- Frequent terms are less informative than rare terms
- Consider a query term that is frequent in the collection (e.g., *high*, *increase*, *line*)
- A document containing such a term is more likely to be relevant than a document that doesn't
- But it's not a sure indicator of relevance.
 - For frequent terms, we want high positive weights for words like *high*, *increase*, and *line*
- But lower weights than for rare terms.
- We will use document frequency (df) to capture this.

TF - IDF

Document 1 : The game of life is a game of everlasting learning

Document 2 : The unexamined life is not worth living

Document 3 : Never stop learning

Query : life learning

Step 1: Term Frequency (TF)

Step 2 :Normalized Term Frequency

Step 3: Inverse Document Frequency (IDF)

Weighted TF * IDF

TF, IDF and weighted TF*IDF for the Query

Step 4: Vector Space Model – Cosine Similarity

Calculate the cosine similarity of the query and Document1, 2, 3

Step1:

Term frequency (TF)

Number of time a word occur in a document.

	The	game	of	life	is	a	everlasting	learning	unexamined	not
DOC1	1	2	2	1	1	1	1	0	0	0
DOC2	1	0	0	1	1	0	0	0	1	1
DOC3	0	0	0	0	0	0	0	1	0	0

	worth	living	never	stop
DOC1	0	0	0	0
DOC2	1	1	0	0
DOC3	0	0	1	1

Normalization of TF needed because in large document the frequency of terms will be much higher than smaller one. So we need to normalize the document based on its size.

Normalized TF =
$$\frac{\text{No. of times the term occurring in the document}}{\text{Total no. of terms in the document}}$$

Normalized TF :

	The	game	of	life	is	a	ever lasting	learning	un examined	not	worth	living	never	stop
DOC1	0.1	0.2	0.2	0.1	0.1	0.1	0.1	0.1						
DOC2	0.142			0.142	0.14				0.142	0.14	0.142	0.142		
DOC3								0.3					0.3	0.3

Inverted Document Frequency

- * Certain terms that occur too frequently have little power, so they have to be weighted down
- * Term occurring less in the document can be more relevant so it has to be weighted up.
- * Logarithm can be used.

Step 2:

$$\text{IDF}(\text{Game}) = 1 + \log_2 \left(\frac{\text{Total no. of documents}}{\text{no. of doc. with term game in it}} \right)$$

$$\text{IDF}(\text{game}) = 1 + \log_2 \left(\frac{3}{1} \right)$$

$$= 1 + 1.0987$$

$$\boxed{\text{IDF}(\text{game}) = 2.098}$$

Terms	IDF
The	1.4
game	2.09
of	2.09
life	1.4
is	1.4
a	2.09
everlasting	2.09
learning	1.4
unexamined	2.09
not	2.09
worth	2.09
living	2.09
Never	2.09
Stop	2.09

Step 3:

Query : life learning

for each term in query = Normalized Term freq \times IDF
in each document

$$\text{Doc 1: life} = 0.1 * 1.40 = 0.140$$

$$\text{learning} = 0.1 * 1.4 = 0.14$$

$$\text{Doc 2: life} = 0.142 * 1.4 = 0.198$$

$$\text{learning} = 0$$

$$\text{Doc 3: life} = 0$$

$$\text{learning} = 0.3 * 1.4 = 0.42$$

Step 4 : Cosine similarity

$$\text{Cosine similarity } (d_1, d_2) = \frac{\text{Dot product } (d_1, d_2)}{| |d_1| * | |d_2| |}$$

$$| |d_1| | = \text{squareroot} \left(d_1[0]^2 + d_1[1]^2 + \dots + d_1[n]^2 \right)$$

$$| |d_2| | = \text{squareroot} \left(d_2[0]^2 + d_2[1]^2 + \dots + d_2[n]^2 \right)$$

Here our search query is (life, learning)

Consider this as a document with 2 terms

$$\text{find } \text{TF}(\text{life}) = \frac{1}{2} = 0.5$$

$$\text{TF}(\text{learning}) = \frac{1}{2} = 0.5$$

Weighted Query

	TF	IDF	TF * IDF
life	0.5	1.40	0.702
learning	0.5	1.40	0.702

Dot product (Query, Document 1) =

(TF * IDF for Query)

$$(0.702) \times (0.140) + (0.702) \times (0.1405)$$

life for doc Learning

$$= 0.1975$$

$$\| \text{Query} \| = \sqrt{(0.7)^2 + (0.7)^2} = 0.99$$

$$\| \text{Document} \| = \sqrt{(0.14)^2 + (0.14)^2} = 0.198$$

$$\text{Cosine similarity} = 0.197 / (0.99) * (0.198)$$

$$= 0.197 / 0.197 = 1.01$$

similarity of query to doc 1 = 1

similarly find for doc 2 & doc 3

Cosine Similarity	DOC1	DOC2	DOC 3
	1	0.707	0.707

Euclidean Distance :

for weighted $TF * IDF$ value, Euclidean distance is measured.

	DOC 1	DOC 2	DOC 3
life (0.702)	0.140	0.200	0
learning (0.702)	0.140	0	0.468

doc1 (0, 140, 0.140)

doc2 (0.20, 0)

doc3 (0, 0.468)

$$\underline{\text{Doc 1}} : \sqrt{(0.70 - 0.14)^2 + (0.70 - 0.14)^2}$$

$$(\text{doc1, Query}) = 0.7916$$

$$\underline{\text{Doc 2}} : \sqrt{(0.702 - 0.200)^2 + (0.702 - 0)^2}$$

$$(\text{doc2, Query}) = 0.803$$

$$\underline{\text{Doc 3}} : \sqrt{(0.702 - 0)^2 + (0.702 - 0.468)^2}$$

$$= 0.7399$$

	Term1	Term2	Term3	Term4	Term6	Total
Doc1	7	2	24	30	5	68
Doc2	0	40	10	3	0	53
Doc3	0	20	30	40	30	120
Doc4	2	3	5	1	0	11
Doc5	4	13	3	33	49	102
Doc6	100	12	2	54	1	168

Query vector = (0, 0, 0, 871, 371)

Find

(10 M)

- 1) TF (ratio of count of a term to total terms in the document)
- 2) IDF
- 3) TFIDF (No need for Vector normalization)

Compare **Cosine similarity** and **Euclidean Distance** way to find the most relevant Document.

	T1	T2	T3	T4	T5
Doc 1	133	20	0	250	50
Doc 2	900	213	0	500	25
Doc 3	50	250	240	0	500
Doc 4	0	20	200	130	40
Doc 5	20	600	50	0	600

Query	{ T5 , T4 }
-------	-------------

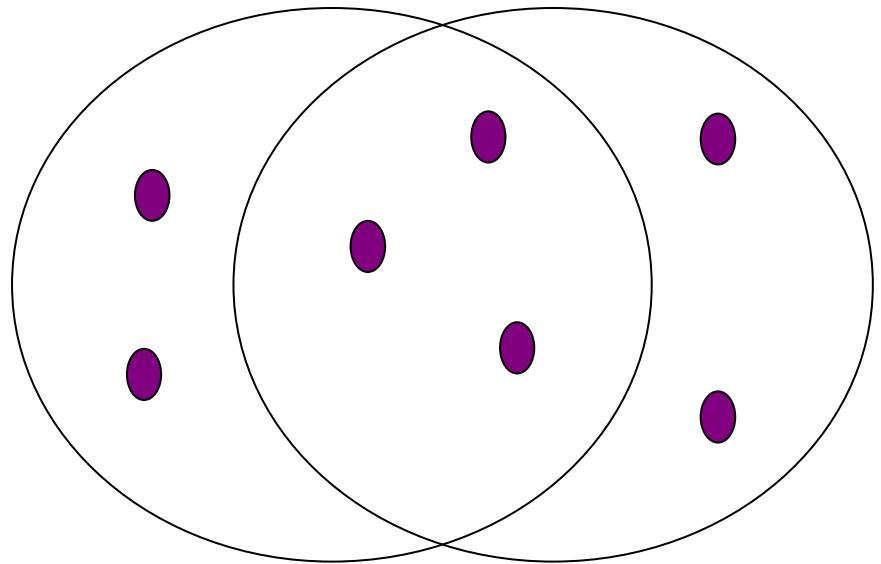
Jaccard Similarity

- The Jaccard Similarity Index can be used to represent the **similarity between two documents**. A **value “0”** means the documents are completely **dissimilar**, **“1”** that they are **identical**, and values between **0 and 1** representing a degree of similarity.

- The **Jaccard similarity** of two **sets** is the size of their intersection divided by the size of their union:

$$sim(C_1, C_2) = |C_1 \cap C_2| / |C_1 \cup C_2|$$

- **Jaccard distance:** $d(C_1, C_2) = 1 - |C_1 \cap C_2| / |C_1 \cup C_2|$



3 in intersection

7 in union

Jaccard similarity= 3/7

Jaccard distance = 4/7

Module - 3

Static and Dynamic Inverted Index

What is Inverted Indices?

- When the crawl is complete, the search engine builds, for **each and every** word, an **inverted index**.
- An inverted index is a list of all documents **containing** that word
 - The index may be a **bit vector**
 - It may also contain the location(s) of the word in the document
- Word: any word in any language, plus misspelling, plus any sequence of characters surrounded by punctuation!
⇒Hundreds of millions of words

Why Inverted Index?

- The basic method of Web search and traditional IR is to find documents that contain the **terms in the user query**.
- Given a user query, one option is to scan the document database sequentially to find the documents that contain the query terms.
- However, this method is obviously impractical for a large collection, such as the Web.
- Another option is to build some **data structures** (called **indices**) from the document collection
- There are many index schemes for this text

Why Inverted Index?

- Efficient **data structures** needed to process large document collections quickly
- How do we store documents in order to maximize **retrieval performance**?
 - We must avoid linear scans of text at query time
 - We must index documents in advance

Dynamic Indexing

- Optimization of an internal file at each time for **retrieval queries**.
- This means that **new terms need to be added to the dictionary**, and postings lists need to be updated for existing terms.
- The simplest way to achieve this is to periodically **reconstruct the index from scratch**.
- This is a good solution if the number of changes over time is small and a ***delay in making new documents searchable is acceptable*** - and if enough resources are available to construct a new index while the old one is still available for querying.

Dynamic Indexing

- If there is a requirement that new documents be included quickly, one solution is to maintain two indexes:
 - ❖ a **large main index**
 - ❖ a **small auxiliary index** that stores new documents.
- The auxiliary index is kept in memory. Searches are run across both indexes and results merged.
- Deletions are stored in an invalidation **bit vector**. We can then filter out deleted documents before returning the search result.
- Documents are **updated by deleting and reinserting them**.
- Each time the **auxiliary index becomes too large**, we **merge it into the main index**.
- The **cost** of this merging operation depends on how we store the index in the file system.

Dynamic Indexing : Need for Compression?

- In this scheme, the reason for keeping the auxiliary index is to reduce the number of disk seeks required over time.
- Unfortunately, the one-file-per-postings-list scheme is infeasible
- A natural solution to this is **index compression**, which aims to represent the same information with fewer bits or bytes

WEB MINING

Module - 3

**Index Compression – Unary, Gamma, Delta &
Golomb Coding**

Need for Compression?

- An inverted index can be **very large**.
- In order to speed up the search, it should reside in memory as much as possible to **avoid disk I/O**.
- Done by reducing the index size and it is an important issue.
- A natural solution to this is **index compression**, which aims to represent the same information with fewer bits or bytes.
- Using compression, the size of an inverted index can be reduced dramatically.
- In the **lossless compression**, the original index can also be reconstructed exactly using the compressed version.
- Since all the information is represented with positive integers, **integer compression** techniques is the main focus.

Compression Types

- There are generally two classes of compression schemes for inverted lists: the **variable-bit scheme** and the **variable-byte scheme**.
- In the variable-bit (also called **bitwise**) scheme, an integer is represented with an **integral number of bits**.
- *Bitwise methods include*
 - **unary coding**
 - **Elias gamma coding**
 - **delta coding**
 - **Golomb coding.**

Compression Types

- In the variable-byte scheme, an integer is stored in an integral number of bytes, where each byte has 8 bits.
- A simple bytewise scheme is the **variable-byte coding**.
- These coding schemes basically map integers onto **self-delimiting binary codewords (bits)**, i.e., the start bit and the end bit of each integer can be detected with no additional delimiters or markers.

Unary Coding

- Unary coding is simple. It represents a number x with $x-1$ bits of zeros followed by a **bit of one**.
- For example, 5 is represented as 00001.
- The one bit is simply the **delimiter**.
- *Decoding is also straightforward.*
- This scheme is effective for very small numbers, but **wasteful for large numbers**.
- It is thus seldom used alone in practice.

Decimal	Unary
1	1
2	01
3	001
4	0001
5	00001
6	000001
7	0000001
8	00000001
9	000000001
10	0000000001

Elias Gamma Coding

Coding:-

- The coding can also be described with the following two steps:
 1. Write x in binary.
 2. Subtract 1 from the number of bits written in step 1 and prepend that many zeros.

Example:-

- The number 9 is represented by 0001001. we first write 9 in binary, which is 1001 with 4 bits, and then prepend three zeros

Elias Gamma Coding

Decimal	Elias Gamma
1	1
2	0 10
3	0 11
4	00 100
5	00 101
6	00 110
7	00 111
8	000 1000
9	000 1001
10	000 1010

Elias Gamma Coding

Decoding:-

We decode an **Elias gamma-coded** integer in two steps:

1. Read and count zeroes from the stream until we reach the first one. Call this count of zeroes K .
2. Consider the one that was reached to be the first digit of the integer, with a value of $2K$, read the remaining K bits of the integer.

Example:-

- ❖ To decompress **0001001**, we first read all zero bits from the beginning until we see a bit of 1.
- ❖ We have $K = 3$ zero bits. We then include the 1 bit with the following 3 bits, which give us 1001.

Elias Gamma Coding

- Gamma coding is efficient for small integers but is not suited to large integers for which the parameterized Golomb code or the Elias delta code is more suitable.

Elias Delta Coding

Coding:-

- In the Elias delta coding, a positive integer x is stored with the gamma code representation of $1+\lfloor \log_2 x \rfloor$, followed by the binary representation of x less the most significant bit.

Example:-

- Let us code the number 9. Since $1+\lfloor \log_2 9 \rfloor = 4$, we have its gamma code 00100 for 4. Since 9's binary representation less the most significant bit is 001, we have the delta code of 001 00001 for 9.

Elias Delta Coding

Selector	Offset
Gamma	Binary

- 1) Gamma Code representation of $1 + \lfloor \log_2 x \rfloor$
(selector)
- 2) Binary representation of 'x' less the MSB
(offset)

Example:

$$x = 9$$

① $1 + \lfloor \log_2 9 \rfloor = 1 + 3 = 4$ in Gamma code

4 in Gamma code

1) $1 + \log_2 x$ in unary

$1 + 2$ in unary
= 001

2) Write x in binary

$4 \Rightarrow \begin{matrix} 1 \\ 0 \\ 0 \end{matrix}$
remove MSB = 00

001	00
-----	----

② Binary of 9 = 1001

less MSB $\times 1001$ ← offset

00100	001
-------	-----

Elias Delta Coding

Decimal	Elias Delta
1	1
2	0 100
3	0 101
4	0 1100
5	0 1101
6	0 1110
7	0 1111
8	00 100000
9	00 100001
10	00 100010

Elias Delta Coding

- An equivalent way to express the same process:
 - Separate X into the highest power of 2 it contains (2^N) and the remaining N binary digits.
 - Encode $N+1$ with Elias gamma coding.
 - Append the remaining N binary digits to this representation of $N+1$.

Number	N	$N+1$	δ encoding
$1 = 2^0$	0	1	1
$2 = 2^1 + 0$	1	2	0 1 0 0
$3 = 2^1 + 1$	1	2	0 1 0 1
$4 = 2^2 + 0$	2	3	0 1 1 00
$5 = 2^2 + 1$	2	3	0 1 1 01
$6 = 2^2 + 2$	2	3	0 1 1 10
$7 = 2^2 + 3$	2	3	0 1 1 11
$8 = 2^3 + 0$	3	4	00 1 00 000
$9 = 2^3 + 1$	3	4	00 1 00 001
$10 = 2^3 + 2$	3	4	00 1 00 010

Elias Delta Coding

Decoding:-

Specifically, we use the following steps:

1. Read and count zeroes from the stream until you reach the first one. **Call this count of zeroes L .**
2. Considering the one that was reached to be the first bit of an integer, with a value of $2L$, read the remaining L digits of the integer. **This is the integer M .**
3. Put a one in the first place of our final output, representing the value $2M$. Read and append the following $M-1$ bits.

Elias Delta Coding

Decoding:-

Example:-

- We want to decode 00100001. We can see that $L = 2$ after step 1, and after step 2, we have read and consumed 5 bits.
- We also obtain $M = 4$ (100 in binary).
- Finally, we prepend 1 to the $M-1$ bits (which is 001) to give 1001, which is 9 in binary.

Decoding

- (ie) L
- 1) Read and count zero's until one reached
 - 2) Read ' L ' bits next from the input
 - 3) Convert it to decimal and get the number of bits (M) $(M-1)$ read from the end.
 - 4) Prepend a '1'

Example:

00100001

- 1) Read till '1' $L = 2$
- 2) Read from '1' ' L ' bits = 100
- 3) Now read $(M-1)$ bits from LSB
and prepend '1'
 $M = 4$

001 from LSB

prepend 1 $\Rightarrow 1001 = 9$

Golomb Coding

The Golomb coding is a form of **parameterized coding** in which integers to be coded are stored as **values relative to a constant b** .

Coding:-

A positive integer x is represented in two parts:

1. The first part is a unary representation of $q+1$, where q is the quotient $\lfloor (x/b) \rfloor$, and
 2. The second part is a special binary representation of the remainder $r = x - qb$. Note that there are b possible remainders.
- For example, if $b = 3$, the possible remainders will be 0, 1, and 2.

Golomb Coding

- write the first few remainders using $\lfloor \log_2 b \rfloor$ bits and the rest using $\lceil \log_2 b \rceil$ bits.
- We must do so such that the decoder knows when $\lfloor \log_2 b \rfloor$ bits are used and when $\lceil \log_2 b \rceil$ bits are used

Golomb Coding : Example

- Let $i = \lfloor \log_2 b \rfloor$. We code the first d remainders using i bits, $d = 2^i + 1 - b$.
- For $b = 3$, to code $x = 9$, we have the quotient $q = \lfloor 9/3 \rfloor = 3$.
- For remainder, we have $i = \lfloor \log_2 3 \rfloor = 1$ and $d = 1$.
- Note that for $b = 3$, there are three remainders, i.e., 0, 1, and 2, which are coded as 0, 10, and 11 respectively. T
- The remainder for 9 is $r = 9 - 3 \times 3 = 0$.
- The final code for 9 is 00010.
- We can see that the first d remainders are standard binary codes, but the rest are not. They are generated using a tree instead.

Coding for $b=3$ and $b=5$

Decimal	Unary	Elias Gamma	Elias Delta	Golomb ($b = 3$)	Golomb ($b = 10$)	Variable byte
1	1	1	1	1 10	1 001	0000001 0
2	01	0 10	0 100	1 11	1 010	0000010 0
3	001	0 11	0 101	01 0	1 011	0000011 0
4	0001	00 100	0 1100	01 10	1 100	0000100 0
5	00001	00 101	0 1101	01 11	1 101	0000101 0
6	000001	00 110	0 1110	001 0	1 1100	0000110 0
7	0000001	00 111	0 1111	001 10	1 1101	0000111 0
8	00000001	000 1000	00 100000	001 11	1 1110	0001000 0
9	000000001	000 1001	00 100001	0001 0	1 1111	0001001 0
10	0000000001	000 1010	00 100010	0001 10	01 000	0001010 0

Decoding

Decoding: To decode a Golomb-coded integer x , we use the following steps:

1. Decode unary-coded quotient q (the relevant bits are consumed).
2. Compute $i = \lfloor \log_2 b \rfloor$ and $d = 2^{i+1} - b$.
3. Retrieve the next i bits and assign it to r .
4. If $r \geq d$ then
 - retrieve one more bit and append it to r at the end;
 - $r = r - d$.
5. Return $x = qb + r$.

Some explanation is in order for step 4. As we discussed above, if $r \geq d$ we need $i+1$ bits to code the remainder. The first line of step 4 retrieves the additional bit and appends it to r . The second line obtains the true value of the remainder r .

Variable-Byte Coding

- In this method, seven bits in each byte are used to code an integer, with the least significant bit set to 0 in the last byte, or to 1 if further bytes follow.
- In this way, small integers are represented efficiently.
- For example, 135 is represented in two bytes, since it lies in the range 27 and 214, as 00000011 00001110.

Variable Byte Coding

Decimal	Unary	Elias Gamma	Elias Delta	Golomb ($b = 3$)	Golomb ($b = 10$)	Variable byte
1	1	1	1	1 10	1 001	0000001 0
2	01	0 10	0 100	1 11	1 010	0000010 0
3	001	0 11	0 101	01 0	1 011	0000011 0
4	0001	00 100	0 1100	01 10	1 100	0000100 0
5	00001	00 101	0 1101	01 11	1 101	0000101 0
6	000001	00 110	0 1110	001 0	1 1100	0000110 0
7	0000001	00 111	0 1111	001 10	1 1101	0000111 0
8	00000001	000 1000	00 100000	001 11	1 1110	0001000 0
9	000000001	000 1001	00 100001	0001 0	1 1111	0001001 0
10	0000000001	000 1010	00 100010	0001 10	01 000	0001010 0

Variable Length Decoding

Decoding is performed in two steps:

1. Read all bytes until a byte with the zero last bit is seen.
2. Remove the least significant bit from each byte read so far and concatenate the remaining bits.

For example,
is decoded to 0000001**1** 0000111**0**
 0000001 0000111

Latent Semantic Indexing

- The retrieval models discussed so far are based on keyword or term matching, i.e., matching terms in the user query with those in the documents.
- However, many concepts or objects can be described in multiple ways (using different words) due to the context and people's language habits.
- If a user query uses different words from the words used in a document, the document will not be retrieved although it may be relevant because the document uses some synonym's of the words in the user query.
- This causes low recall. For example, “picture”, “image” and “photo” are **synonyms** in the context of digital cameras.
- If the user query only has the word “picture”, relevant documents that contain “image” or “photo” but not “picture” will not be retrieved.

Latent Semantic Indexing

- Latent semantic indexing (LSI), proposed by Deerwester aims to deal with this problem through the identification of statistical associations of terms.
- It then uses a statistical technique, called **singular value decomposition** (SVD), to estimate this latent structure, and to remove the “noise”.
- The results of this decomposition are descriptions of terms and documents based on the latent semantic structure derived from SVD.
- This structure is also called the **hidden “concept” space**, which associates syntactically different but semantically similar terms and documents.

Latent Semantic Indexing

- Let D be the text collection, the number of distinctive words in D be m and the number of documents in D be n .
- LSI starts with an mxn term document matrix \mathbf{A} .
- Each row of \mathbf{A} represents a term and each column represents a document.
- The matrix may be computed in various ways, e.g.,
- using term frequency or TF-IDF values.
- We use term frequency as an example in this section.
- Thus, each entry or cell of the matrix \mathbf{A} , denoted by A_{ij} , is the number of times that term i occurs in document j .

Singular Value Decomposition

- What SVD does is to factor matrix A (a $m \times n$ matrix) into the product of three matrices, i.e.,

$$A = U \Sigma V^T$$

where

U is a $m \times r$ matrix and its columns, called **left singular vectors**, are eigenvectors associated with the r non-zero eigenvalues of AA^T . Furthermore, the columns of U are unit orthogonal vectors, i.e., $U^T U = I$ (identity matrix).

V is an $n \times r$ matrix and its columns, called **right singular vectors**, are eigenvectors associated with the r non-zero eigenvalues of $A^T A$. The columns of V are also unit orthogonal vectors, i.e., $V^T V = I$.

Σ is a $r \times r$ diagonal matrix, $\Sigma = \text{diag}(\sigma_1, \sigma_2, \dots, \sigma_r)$, $\sigma_i > 0$. $\sigma_1, \sigma_2, \dots$, and σ_r , called **singular values**, are the non-negative square roots of the r (non-zero) eigenvalues of AA^T . They are arranged in decreasing order, i.e., $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_r > 0$.