Web Search Engines

→ Crawler
- Simple Crawler
- Large-Scale Crawler
- Efficient DNS Resolution
- Robot Exclusion Protocol
- (Near-)Duplicate Detection
- Indexer
- Query Interface
- Ranker
- Scalability

Web search engines

- Rooted in Information Retrieval (IR) systems
  - Prepare a keyword index for corpus
  - Respond to keyword queries with a ranked list of documents.

- ARCHIE
  - Earliest application of rudimentary IR systems to the Internet
  - Title search across sites serving files over FTP

Search Engines

http://www.searchenginewatch.com

- **Crawler**
  - collect internet addresses
- **Indexer**
  - break up text into tokens (words)
  - create inverted index
  - advanced indices include position information and hyperlink information
- **Query interface**
  - query for words and phrases
  - Boolean expressions
  - search for location, site, url, domain, etc.
- **Ranker**
  - heuristics based on frequency/location of words
  - heuristics based on hyperlink structure (page rank (Google))
  - pre-defined categories or clustering of results

Crawling and indexing

- Purpose of crawling and indexing
  - quick fetching of large number of Web pages into a local repository
  - indexing based on keywords
  - Ordering responses to maximize user's chances of the first few responses satisfying his information need.

- Earliest search engine:
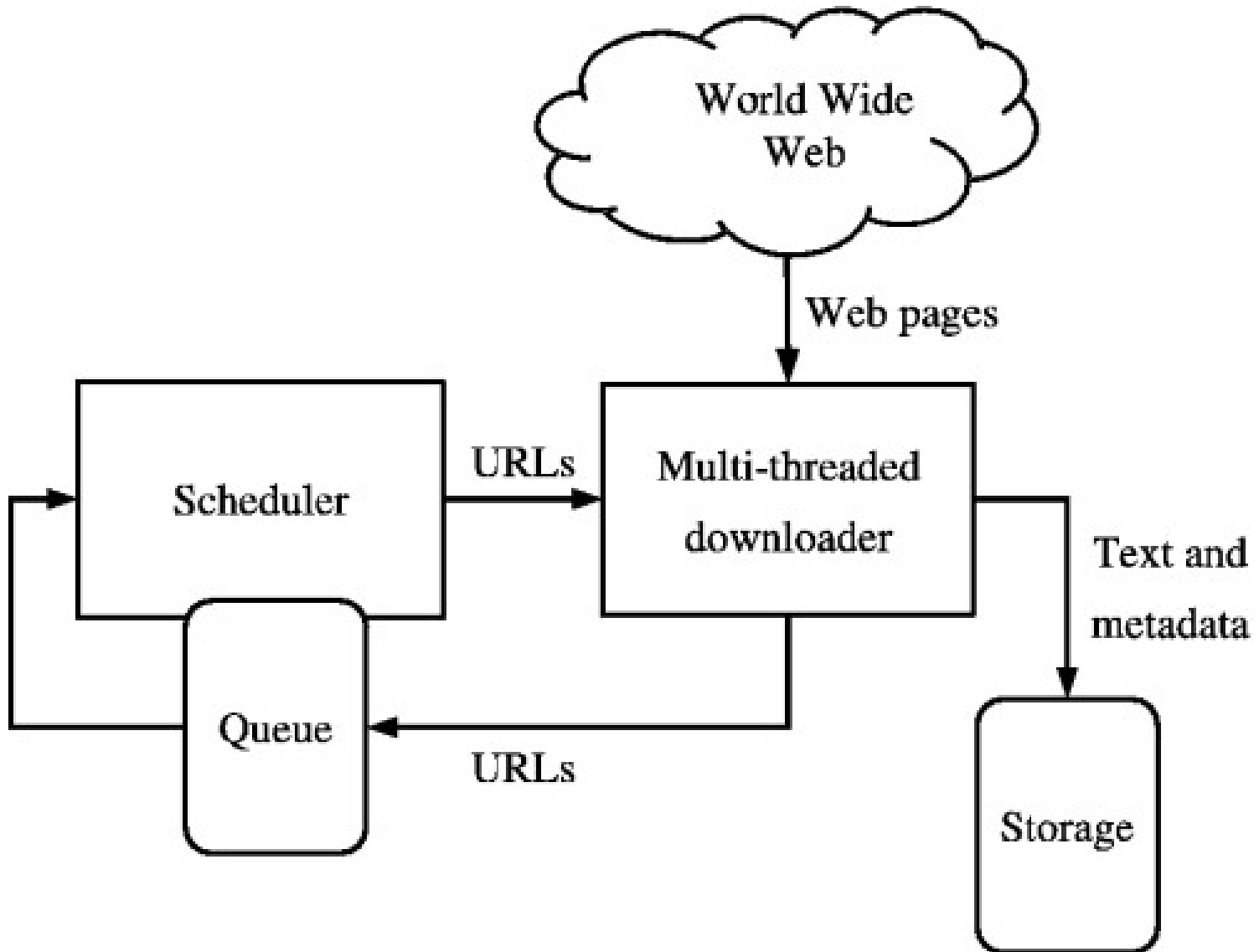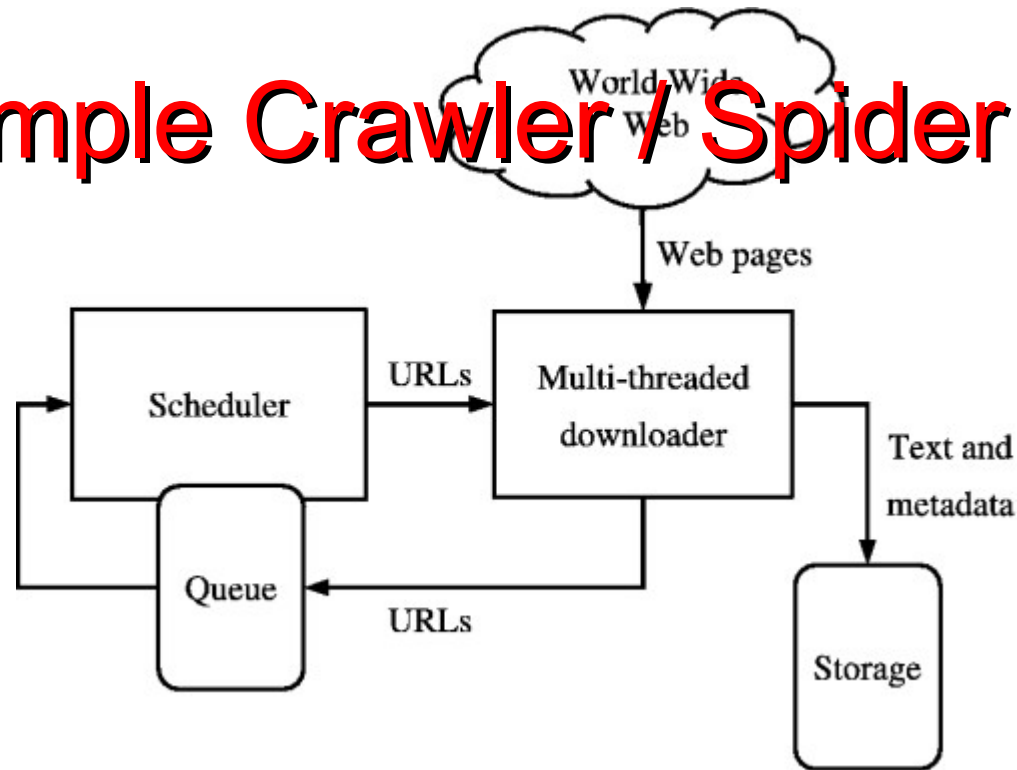 - Lycos (Jan 1994)

- Followed by….
  - Alta Vista (1995), HotBot and Inktomi, Excite

Simple Crawler / Spider

Simple Crawler / Spider

1. Initialize Queue with a (set of) random starting URL(s)
2. retrieve the first URL in the Queue
3. find all hyperlinks in the retrieved page
4. add new hyperlinks to the Queue (remove duplicates)
5. store retrieved page
6. goto 2.

J. Fürnkranz

Crawling procedure

- Simple Crawlers are easy

- But a great deal of engineering goes into industry-strength crawlers
    - Industry crawlers crawl a substantial fraction of the Web
    - many times (as quickly as possible)
    - E.g.: Google, AltaVista, Yahoo! (bought Inktomi)

- 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.
    - usually stopped after a certain amount of work

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

- (Logical) 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.
  - a crawler never completes it job, it is simply stopped (and re-started)

Typical Anatomy of a Large-Scale Crawler
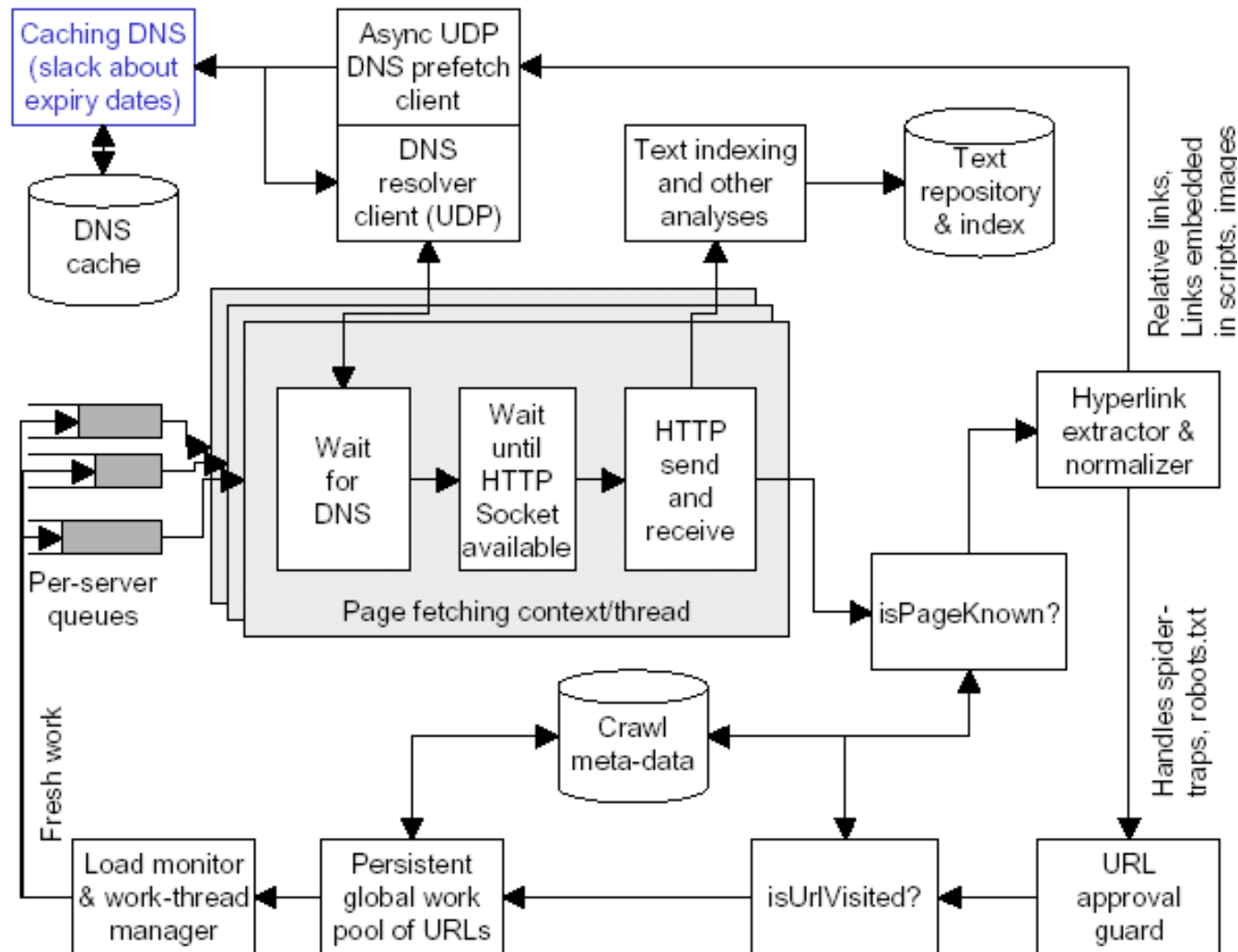
Normalizing URLs

- URLs may appear in many forms
  - absolute vs. relative links (e.g., „`../photo.jpg`")
  - with or without ports, etc.
- To recognize duplicates, a Canonical URL is formed by
  - Using a standard string for the protocol
    - e.g., `http` vs. `Http`
  - Canonicalizing the host name
    - lower case
    - resolving relative URLs
  - Adding an explicit port number (default: 80)
  - Normalizing and cleaning up the path
    - e.g., removing `/./` or `/xxx/../` from path

DNS Caching

- A crawler will spend a substantial amount of time in resolving DNS requests

- Crucial performance enhancements
  - A large, persistent DNS cache
    - need not be super-fast (disk/network are the bottlenecks)
  - Custom client for DNS name resolution
    - allows to concurrently handle multiple outstanding requests
    - issue them and poll at a later time for completion
    - distribute load among multiple DNS servers
    - Compaq Mercator crawler reduced time spent in DNS resolution from 87% to 25% by using a custom crawler
  - Prefetching
    - immediately after URL is extracted, the domain-host is sent to a pre-fetching client that makes sure the address is in the cache.

J. Fürnkranz

Per-server work queues

- http servers protect against Denial Of Service (DoS) attacks
  - limit the speed or frequency of responses to any fixed client IP address
- Avoiding DOS
  - limit the number of active requests to a given server IP address at any time
  - maintain a queue of requests for each server
    - supported by HTTP/1.1 persistent socket capability
  - distribute attention relatively evenly between a large number of sites
- Access locality vs. politeness dilemma

Robot exclusion

- Check
  - whether the server prohibits crawling a normalized URL
  - In robots.txt file in the HTTP root directory of the server
    - species a list of path prefixes which crawlers should not attempt to fetch.
- Meant for crawlers only
- Examples:
  - `http://www.google.de/robots.txt`
  - `http://www.fleiner.com/robots.txt`

Spider traps

Protecting from crashing on

- Ill-formed HTML
  - E.g.: page with 68 kB of null characters
- Misleading sites
  - indefinite number of pages dynamically generated by CGI scripts
  - paths of arbitrary depth created using
    - soft directory links and
    - path remapping features in HTTP server
  - e.g., http://www.fleiner.com/bots
    http://www.fleiner.com/botsv/

16

Spider Traps: Solutions

- Guards
    - Preparing regular crawl statistics
        - Adding dominating sites to guard module
    - Disable crawling active content such as CGI form queries
    - Eliminate URLs with non-textual data types
    - Monitor URL length
    -

- No automatic technique can be foolproof

Eliminating already-visited URLs

- Checking if a URL has already been fetched
  - Before adding a new URL to the work pool
  - Needs to be very quick.
  - Achieved by computing MD5 hash function on the URL
    - 32 – 128 bit signature (depending on size of crawling task)
- Exploiting spatio-temporal locality of access
  - Two-level hash function.
    - most significant bits (say, 24) derived by hashing the host name
    - lower order bits (say, 40) derived by hashing the path
  - concatenated bits used as a key in a B-tree
    - thus spatio-temporal locality is maintained
- new URLs added to frontier of the crawl.
  - hash values added to B-tree.

Avoiding duplicate pages

- Reduce redundancy in crawls
- Duplicate detection
  - Identify Mirrored Web pages and sites
  1. Detecting exact duplicates via hyperlink information
     - Checking against MD5 digests of stored URLs
     - Representing a relative out-link v (relative to pages u_1 and u_2) as tuples $(\mathrm{hash}(u_1); v)$ and $(\mathrm{hash}(u_2); v)$
  2. Detecting near-duplicates based on text
     - Hard problem: Even a single altered character will completely change the digest !
     - E.g.: date of update, name and email of the site administrator

Shingling

- Automated detection of near-duplicate pages
  - Typically during indexing
- shingle (n-gram)
  - sequence of n sucessive words
    - in practice, $n = 10$ has been found to be useful
- assumption:
  - overlap in *sets* of words only indicates similar topic
  - But overlap in *sequences* of words (n-grams) indicates identity
- compute a similarity in terms of shingles using the Jaccard co-efficient

$$r'(d_1, d_2) = \frac{|S(d_1) \cap S(d_2)|}{|S(d_1) \cup S(d_2)|}$$

  - can be approximated efficiently by not computing all shingles
    - e.g., eliminating frequent words that occur in almost all documents

J. Fürnkranz

Determining page changes

- High variance of rate of page changes
- „If-modified-since" request header with HTTP protocol
  - Impractical for a crawler
- "Expires" HTTP response header
  - For pages that come with an expiry date
- Otherwise need to guess if revisiting that page will yield a modified version.
  - Score reflecting probability of page being modified
  - Crawler fetches URLs in decreasing order of score.
  - Assumption on update rate: _recent past predicts the future_
- Small scale intermediate crawler runs
  - to monitor fast changing sites
    - E.g.: current news, weather, etc.
  - Patched intermediate indices into master index

Text repository

- Fetched pages are dumped into a repository
- Decoupling crawler from other functions for efficiency and reliability preferred
  - e.g., building a topic hierarchy, a hyperlink graph, etc.
- Page-related information stored in two parts
  - meta-data
    - includes fields like content-type, last-modified date, content-length, HTTP status code, etc.
  - page contents
    - stored in compressed form
      - often distributed over multiple servers
    - simple access methods for
      - crawler to add pages
      - Subsequent programs (Indexer etc) to retrieve documents

Web Search Engines

- Crawler
→ Indexer
  - Tokenization
  - Document/Term Matrix
  - Inverted Index
  - Index Compression Techniques
    - Sparse Encoding
    - Gap Encoding and Gamma Code
    - Lossy Compression Techniques
- Query Interface
- Ranker
- Scalability

J. Fürnkranz

Document preprocessing: Tokenization

- Filter textual parts that are not meant to be indexed
  - tags
  - Optional:
    - stop-word removal
    - stemming/conflation of words

  > will be covered later

- Tokens
  - regarded as nonempty sequence of characters excluding spaces and punctuations.
  - represented by a suitable integer, tid, typically 32 bits
- Result of Tokenization
  - document (did) transformed into a sequence of integers (tid, pos)

A Document is a Bag of Words
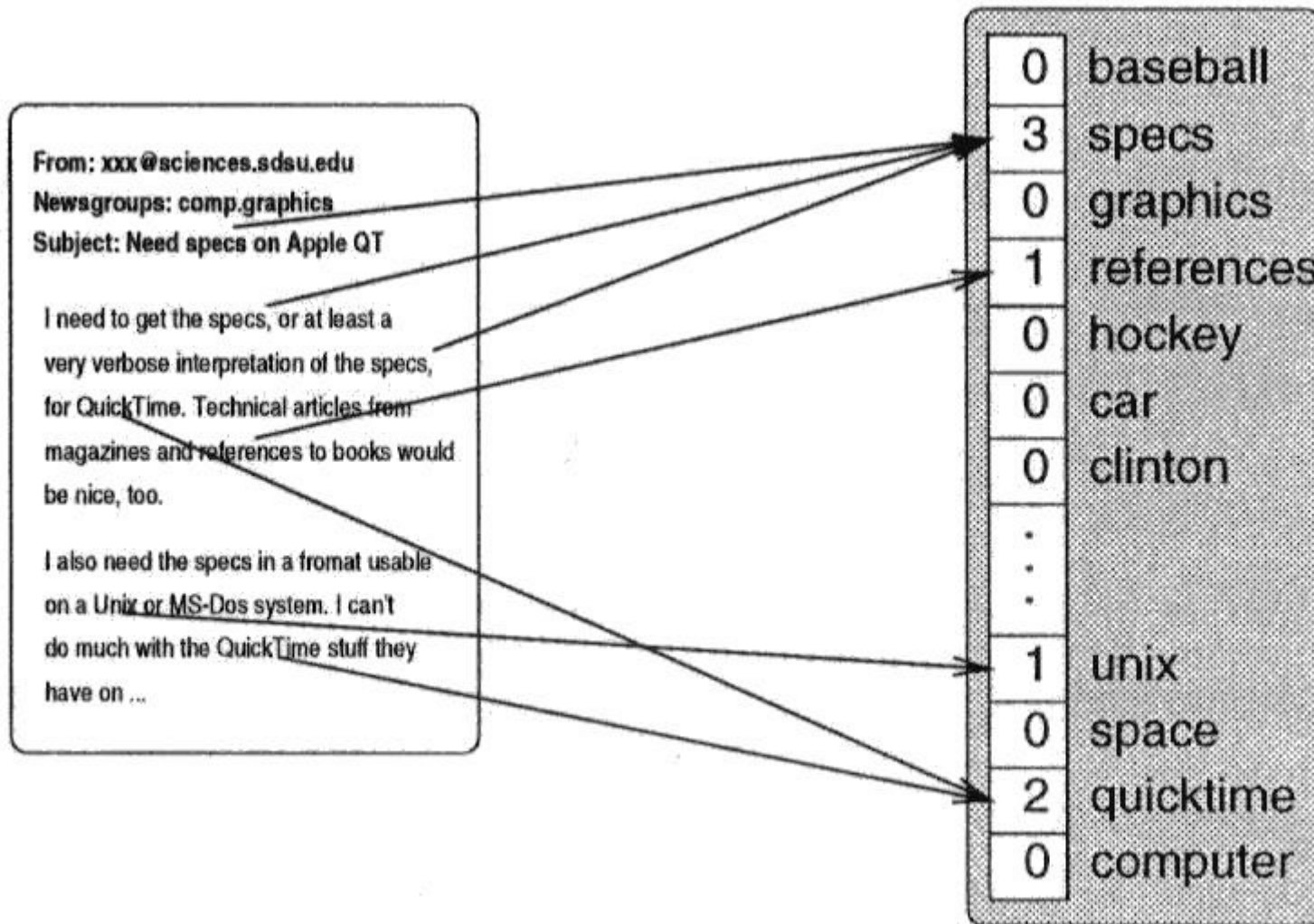
Document / Term Matrix

- A collection of documents can be represented as a matrix
  - **ROWS:** documents
  - **COLUMNS:** feature values

	baseball	specs	graphics	....	quicktime	computer
D1	0	3	0	....	2	0
D2	1	2	0	...	0	0
D3	0	0	2	...	1	5
.....	....	....	....	....	....	....

Inverted Index

- To optimize retrieval generate an *inverted index*

- This is the document/term matrix transposed
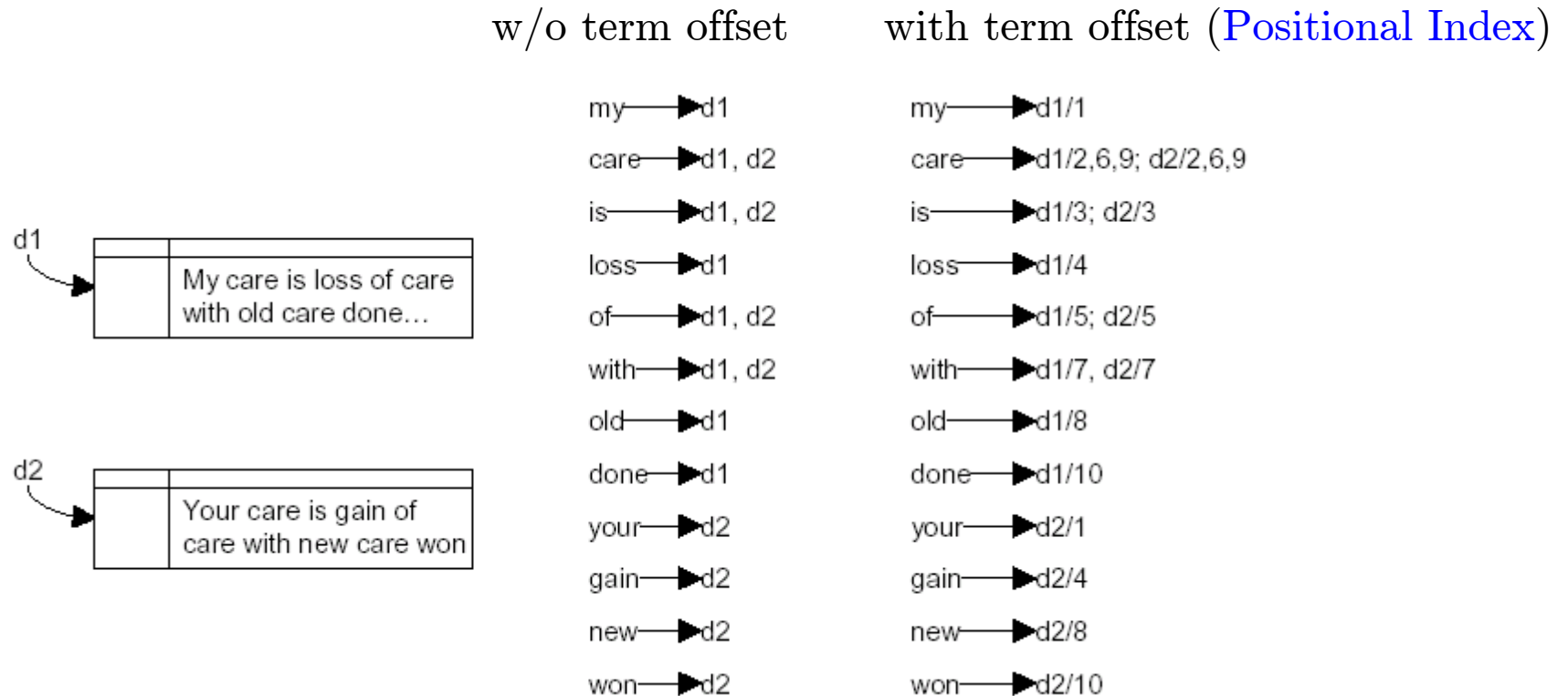
- facilitates efficient look-up of query term

	D1	D2	D3
baseball	0	1	0
specs	3	2	0
graphics	0	0	2
.....	....	....	....
quicktime	2	0	1
computer	0	0	5

Sparse Encoding of Documents

- Storing the inverted index is too costly
    - most of the entries will be 0

- Solution
    - store the list of documents associated with each term
    - extensions allow to store additional information
        - location of term in document
        - location of term in paragraph
        - etc.

Sparse Encoding Examples

Two variants of the inverted index data structure, usually stored on disk.

w/o term offset   with term offset (Positional Index)

d1

| | My care is loss of care with old care done… |

d2

| | Your care is gain of care with new care won |

w/o term offset	with term offset
my → d1	my → d1/1
care → d1, d2	care → d1/2,6,9; d2/2,6,9
is → d1, d2	is → d1/3; d2/3
loss → d1	loss → d1/4
of → d1, d2	of → d1/5; d2/5
with → d1, d2	with → d1/7, d2/7
old → d1	old → d1/8
done → d1	done → d1/10
your → d2	your → d2/1
gain → d2	gain → d2/4
new → d2	new → d2/8
won → d2	won → d2/10

Term Offsets: The mapping from terms to documents and positions (written as "document/position") may be implemented using a B-tree or a hash-table.

Size of Positional Index

- We need an entry for each occurrence of a word, not just once per document

  $\rightarrow$ Index size depends on average document size

- Rules of Thumb

  - A positional index is 2–4 times as large as a non-positional index
  - Positional index size 35–50% of volume of original text
  - Caveat: all of this holds for "English-like" languages

Manning and Raghavan   J. Fürnkranz

Index Size Reduction by Filtering

- Stemming/case folding/no numbers cuts may reduce
  - number of terms by ~35%
  - number of list entries by 10-20%

  will be covered later

- Stop words
  - Rule of 30: ~30 words account for ~30% of all term occurrences in written text [= # term offsets]
  - Eliminating 150 commonest terms from index will reduce list entries ~30% *without considering compression*
    - With compression, you save ~10%

Manning and Raghavan   J. Fürnkranz

Index compression techniques

- Compressing the index so that much of it can be held in memory
  - Required for high-performance IR installations (as with Web search engines),
- Redundancy in index storage
  - Storage of document IDs.

Delta encoding or **Gap Encoding**

- Sort Doc IDs in increasing order
- Store the first ID in full
- Subsequently store only difference *(gap)* from previous ID
- Example:
  - *word* appears in documents (10000, 10030, 10100)
  - *word* appears in documents (10000, +30, +70)

Zipf's Law

- The *k*th most frequent term has frequency proportional to *1/k*.

10th frequent term occurs ca. 10,000 times

most frequent term occurs ca. 100,000 times

J. Fürnkranz
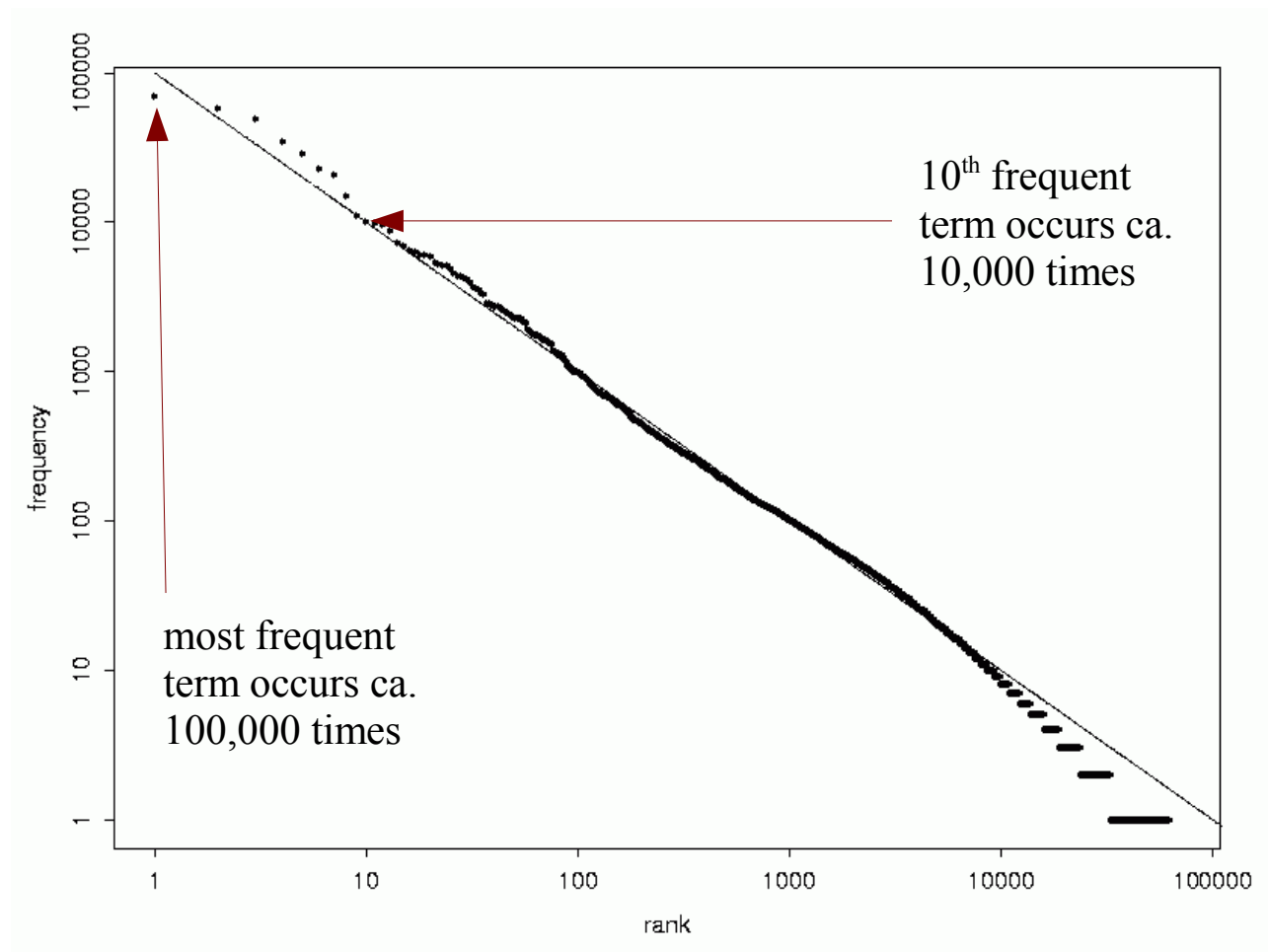
Encoding gaps

Goal: Small gap must cost far fewer bits than a full did.

- **Binary encoding**
  - regular encoding for integers
  - Optimal when all symbols are equally likely
- **Unary code**
  - the number n is represented with n consecutive 0's followed by a 1 (or, conversely, consecutive 1's followed by a 0)
  - optimal if probability of gaps of size n decays exponentially

  $$(Pr(n)=2^{-n})$$
- **Gamma code**
  - Represent gap x as
    - **Order of Magnitude:** Unary code for $1+\lfloor \log x \rfloor$ followed by
    - **Exact Value:** $x-2^{\lfloor \log x \rfloor}$ represented in binary ($\lfloor \log x \rfloor$ bits)
- **Golomb codes**
  - Further enhancement

Gamma Coding

1. Separate the integer into the highest power of 2 it contains (2^N) and the remaining N binary digits of the integer.

2. Encode N in unary; that is, as N zeroes followed by a one (which may also be viewed as the first digit, representing 2^N)

3. Append the remaining N binary digits to this representation of N.

```
                                           Implied probability
 1 = 2⁰ + 0 = 1                                 1/2
 2 = 2¹ + 0 = 010                               1/8
 3 = 2¹ + 1 = 011                                "
 4 = 2² + 0 = 00100                             1/32
 5 = 2² + 1 = 00101                              "
 6 = 2² + 2 = 00110                              "
 7 = 2² + 3 = 00111                              "
 8 = 2³ + 0 = 0001000                           1/128
 9 = 2³ + 1 = 0001001                            "
10 = 2³ + 2 = 0001010                            "
11 = 2³ + 3 = 0001011                            "
12 = 2³ + 4 = 0001100                            "
13 = 2³ + 5 = 0001101                            "
14 = 2³ + 6 = 0001110                            "
15 = 2³ + 7 = 0001111                            "
16 = 2⁴ + 0 = 000010000                         1/512
17 = 2⁴ + 1 = 000010001                          "
...
```

Wikipedia

J. Fürnkranz

Lossy compression mechanisms

- one does not need to identify the exact document
  - identify group of documents and then search for the right one
- → collect documents into buckets
  - Construct inverted index from terms to bucket IDs
  - Document IDs shrink to half their size.
- Cost: time overheads
  - For each query, all documents in that bucket need to be scanned
    - Trading off space for time
  - Solution: index documents in each bucket separately
- the same technique can also be used for encoding positions in documents
  - index block IDs instead of position IDs (*block addressing*)

Indexing Phrases

- Including phrases to rank complex queries
  - Operators to specify word inclusions and exclusions
  - With operators and phrases queries/documents can no longer be treated as ordinary points in vector space

- Dictionary of phrases
  - Could be catalogued manually
  - Could be derived from the corpus itself using statistical techniques
  - Two separate indices:
    - one for single terms and another for phrases

will be covered later

Other issues

- Spamming
    - Adding popular query terms to a page unrelated to those terms
        - E.g.: Adding "Hawaii vacation rental" to a page about "Internet gambling"
    - Little setback due to hyperlink-based ranking (now we have link-spam...)

- Titles, headings, meta tags and anchor-text
    - TFIDF framework treats all terms the same
    - Meta search engines:
        - Assign weight age to text occurring in tags, meta-tags
    - Using anchor-text on pages u which link to v
        - Anchor-text on u offers valuable information about v as well.

Web Search Engines

- Crawler
- Indexer
- Query Interface
    - Simple Boolean Queries
    - Efficient Processing
        - Sparse Encoding
        - Skip Pointers
    - Advanced Processing
- Ranker
- Scalability

Simple Boolean Queries

look up the inverted index

- **One-term queries** (`T1`):

  $\Rightarrow$ look up term, return documents with non-zero entries

- **Conjunctive queries** (`T1 AND T2`):

  $\Rightarrow$ Intersection of documents with non-zero entries

- **Disjunctive queries** (`T1 OR T2`):

  $\Rightarrow$ Union of documents with non-zero entries

- **Negation** (`NOT T1`):

  $\Rightarrow$ documents with zero entries

	D1	D2	D3
baseball	0	1	0
specs	3	2	0
graphics	0	0	2
.....	....	....	....
quicktime	2	0	1
computer	0	0	5

Boolean Queries with List Representation

- Walk through the two postings simultaneously, in time linear in the total number of postings entries

- Example:

Query
"Brutus AND Caesar"
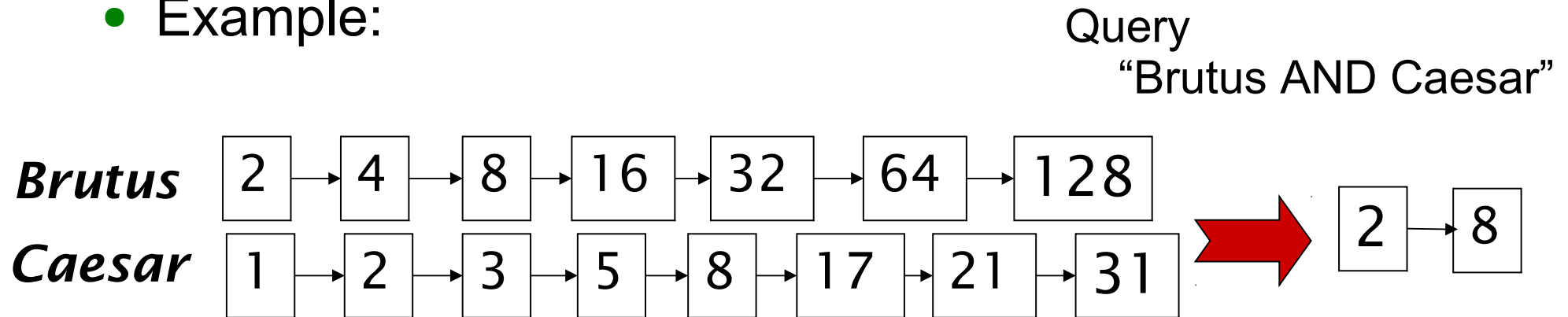
If the list lengths are *m* and *n*, the merge takes O(*m+n*) operations.

Can we do better?
Yes, if index isn't changing too fast.

                J. Fürnkranz

Augment Lists with Skip Pointers
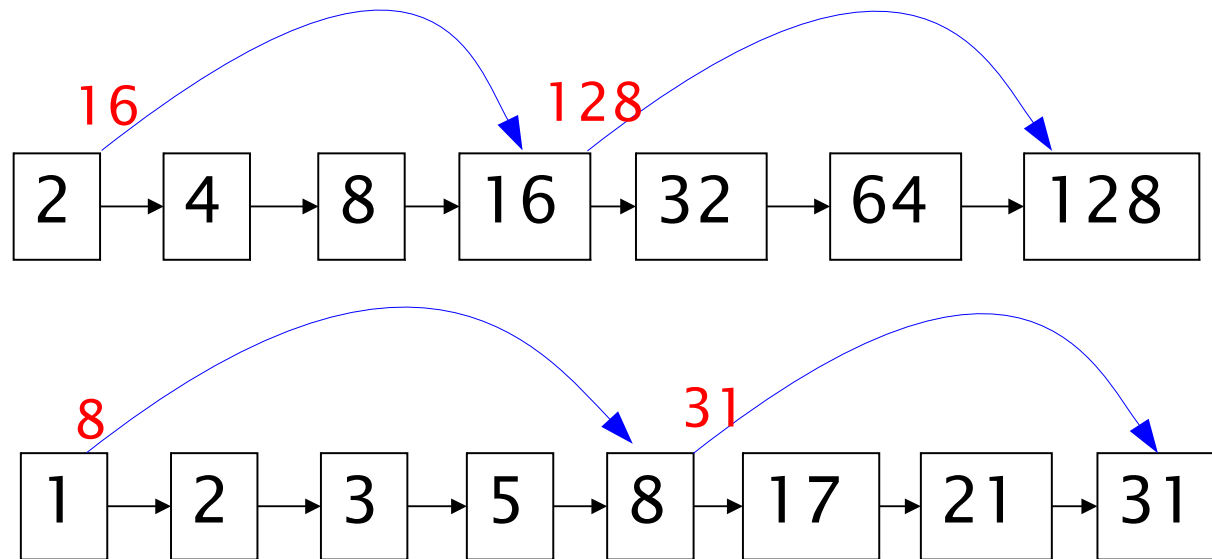
- Why?
  - To quickly skip over positions that will not appear in the search result.
- How?
- Where do we place skip pointers?

Query processing with skip pointers
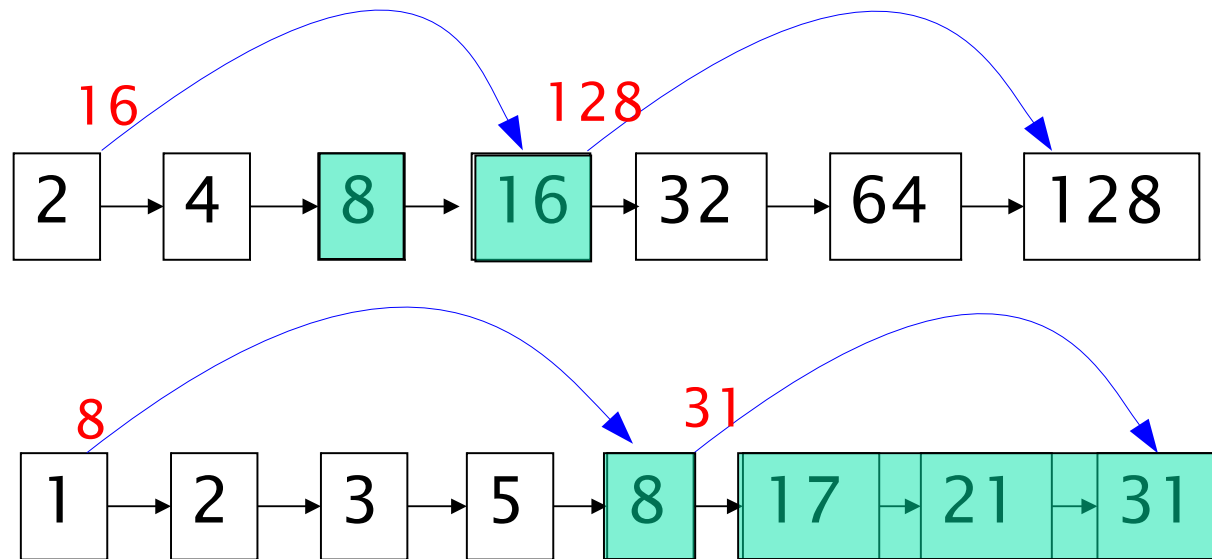
Suppose we've stepped through the lists until we process **8** on each list.

When we get to **16** on the top list, we see that its successor is **32**.

But the skip successor of **8** on the lower list is **31**, so we can skip ahead past the intervening postings.

Manning and Raghavan   J. Fürnkranz

Where do we place skips?

- Tradeoff:
    - More skips $\rightarrow$ shorter skip spans $\Rightarrow$ more likely to skip. But lots of comparisons to skip pointers.
    - Fewer skips $\rightarrow$ few pointer comparison, but then long skip spans $\Rightarrow$ few successful skips.
- Simple heuristic:
    - for lists of length L, use $\sqrt{L}$ evenly spaced skip pointers.
    - This ignores the distribution of query terms.
    - Easy if the index is relatively static; harder if L keeps changing because of updates.
- This definitely used to help; with modern hardware it may not (Bahle et al. 2002)
    - The cost of loading a bigger lists outweighs the gain from quicker in-memory merging

Manning and Raghavan   J. Fürnkranz

Advanced Queries

- location of query words in text
  - document title
  - anchor text
- collocations
  - phrases
  - words in proximity
  - words in same sentence/paragraph
- location on Web
  - restrict domains
  - restrict hosts
- pages that link to a page

Example: Altavista Advanced Search

Example: Altavista Search Syntax

Home › AltaVista Help › Search › **Special search terms**

domain:*domainname*	Finds pages within the specified domain. Use **domain:uk** to find pages from the United Kingdom, or use **domain:com** to find pages from commercial sites.
host:*hostname*	Finds pages on a specific computer. The search **host:www.shopping.com** would find pages on the Shopping.com computer, and **host:dilbert.unitedmedia.com** would find pages on the computer called dilbert at unitedmedia.com.
link:*URLtext*	Finds pages with a link to a page with the specified URL text. Use **link:www.myway.com** to find all pages linking to myway.com.
title:*text*	Finds pages that contain the specified word or phrase in the page title (which appears in the title bar of most browsers). The search **title:sunset** would find pages with sunset in the title.
inurl:*text*	Finds pages with a specific word or phrase in the URL. Use **inurl:garden** to find all pages on all servers that have the word *garden* anywhere in the host name, path, or filename.

Business Services   Submit a Site   About AltaVista   Privacy Policy   Help

© 2007 Overture Services, Inc.

J. Fürnkranz

Web Search Engines

- Crawler
- Indexer
- Query Interface
- → Ranker
  - The Vector-Space Model
  - Similarity-Based Ranking
  - Evaluation of Ranking Results
    - Recall and Precision
    - Recall and Precision Curves
    - (N)DCG
  - Improving Retrieval Efficiency
- Scalability

The Vector Space Model

- Origin:

  Information Retrieval, SMART system (Salton et al.)

- Basic idea:

  - A document is regarded as a vector in an n-dimensional space

    - 1 dimension for each possible word (*feature, token*)
    - the value in each dimension is (in the simplest case) the number of times the word occurs in the document (*term frequency – TF*)

  - a document is a linear combination of the base vectors

  - linear algebra can be used for various computations

Intuition

$$\mathbf{d}_i = (d_{i,1}, d_{i,2}, d_{i,3})$$
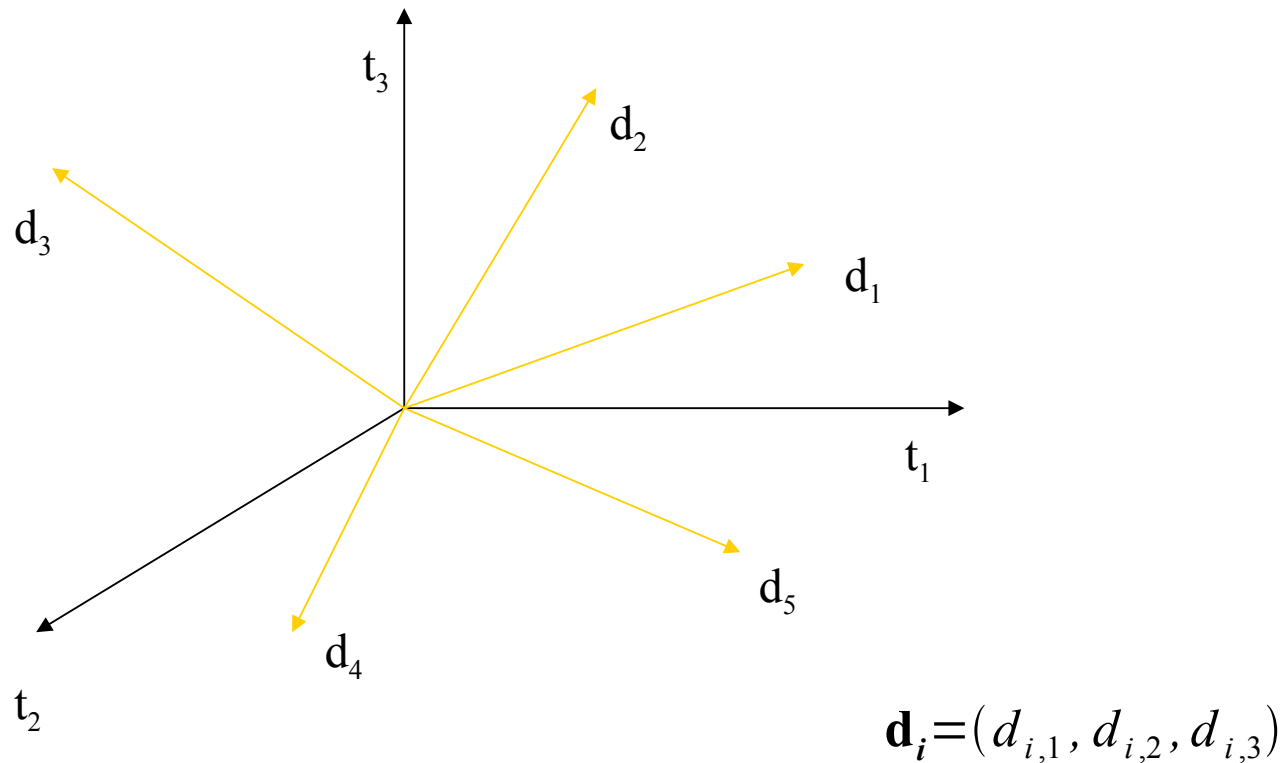
Postulate: Documents that are "close together" in the vector space talk about the same things.

    J. Fürnkranz

Term Weighting

Different ways for computing the $d_{i,j}$:

- **Boolean**

  - possible values are only
    - 0 (term does not occur in document)
    - 1 (term does occur)

$$d_{i,j} = \begin{cases} 0 \text{ if } t_j \notin \mathbf{d}_i \\ 1 \text{ if } t_j \in \mathbf{d}_i \end{cases}$$

- **Term Frequency (TF)**

  - term is weighted with the frequency of its occurrence in the text

$$d_{i,j} = TF(\mathbf{d}_i, t_j)$$

- **Term Frequency - Inverse Document Frequency (TF-IDF)**

  - Idea: A term is characteristic for a document if
    - it occurs frequently in this document (TF)
    - occurs infrequently in other documents (IDF)
  - divides TF by DF
    (or multiplies TF with IDF)

$$d_{i,j} = \frac{TF(\mathbf{d}_i, t_j)}{DF(t_j)} = TF(\mathbf{d}_i, t_j) \cdot IDF(t_j)$$

Term frequency

$$d_{i,j} = TF(\mathbf{d}_i, t_j)$$

- Measures the frequency of the occurrence of a term t in the document $\mathbf{d}$

$$TF(\mathbf{d}, t) = n(\mathbf{d}, t)$$

- Common modifications:

  - normalization with document length (relative frequency)

$$TF(\mathbf{d}, t) = \frac{n(\mathbf{d}, t)}{\underbrace{\sum_\tau n(\mathbf{d}, \tau)}_{\text{document length}}}$$

  - normalization with maximum frequency

$$TF(\mathbf{d}, t) = \frac{n(\mathbf{d}, t)}{\max_\tau n(\mathbf{d}, \tau)}$$

  - logarithmic scaling

$$TF(\mathbf{d}, t) = \log(1 + n(\mathbf{d}, t))$$

  - Cornell SMART system

$$TF(\mathbf{d}, t) = \begin{cases} 0 & \text{if } t \notin \mathbf{d} \\ 1 + \log(1 + \log n(\mathbf{d}, t)) & \text{if } t \in \mathbf{d} \end{cases}$$

Inverse document frequency

- Measure the „rareness" of a word by counting in how many documents it occurs

- Given
    - D is the document collection
    - D_t is the set of documents containing t

- Formulae
    - mostly dampened functions of $IDF(t) = \dfrac{|D|}{|D_t|}$
    - e.g., in the SMART retrieval system $IDF(t) = \log\left(\dfrac{1+|D|}{|D_t|}\right)$

- used for term weighting together with term frequency

$$d_{i,j} = \frac{TF(\mathbf{d}_i, t_j)}{DF(t_j)} = TF(\mathbf{d}_i, t_j) \cdot IDF(t_j)$$

Relevance ranking

- Keyword queries
  - In natural language
  - Queries are not precise
    - entire set of matching documents for response unacceptable
  - Solution
    - Rate each document for how likely it is to satisfy the user's information need (relevance)
    - Sort in decreasing order of the score
    - Present results in a ranked list.

- No algorithmic way of ensuring that the ranking strategy always favors the information need
  - Query: only a part of the user's information need

Similarity of Document Vectors

- First Idea:
  - Distance between $\mathbf{d}_1$ and $\mathbf{d}_2$ is the length of the vector $|\mathbf{d}_1 - \mathbf{d}_2|$ (measured with Euclidean distance)

- Why is this not a great idea?
  - Short documents would be more similar to each other by virtue of length, not topic

  $\rightarrow$ We have to deal with the issue of length normalization

  - explicit normalization (as, e.g., through normalized TF)

- Alternative proposal:
  - We can also implicitly normalize by looking at *angles* between document vectors instead

Manning and Raghavan   J. Fürnkranz

Cosine similarity

- Distance between vectors $\mathbf{d}_1$ and $\mathbf{d}_2$ *captured* by the cosine of the angle θ between them.

Cosine similarity

- Distance between vectors $\mathbf{d}_1$ and $\mathbf{d}_2$ *captured* by the cosine of the angle θ between them.

$$\cos(\theta) = \frac{\mathbf{d}_1 \cdot \mathbf{d}_2}{\|\mathbf{d}_1\| \cdot \|\mathbf{d}_2\|} = \frac{\sum_{i=1}^{n} d_{1,i} d_{2,i}}{\sqrt{\sum_{i=1}^{n} d_{1,i}^2} \cdot \sqrt{\sum_{i=1}^{n} d_{2,i}^2}}$$

Cosine similarity

- Distance between vectors $\mathbf{d}_1$ and $\mathbf{d}_2$ *captured* by the cosine of the angle θ between them.
- the distance is invariant to re-scaling the vector
    - e.g., if two copies of document $\mathbf{d}_1$ are concatenated to a new document $\mathbf{d}_3$, the similarity to $\mathbf{d}_2$ remains the same

$$\cos(\theta) = \frac{\mathbf{d}_1 \cdot \mathbf{d}_2}{\|\mathbf{d}_1\| \cdot \|\mathbf{d}_2\|} = \frac{2 \cdot \mathbf{d}_1 \cdot \mathbf{d}_2}{2 \cdot \|\mathbf{d}_1\| \cdot \|\mathbf{d}_2\|} = \frac{\mathbf{d}_3 \cdot \mathbf{d}_2}{\|\mathbf{d}_3\| \cdot \|\mathbf{d}_2\|}$$

because $\|c \cdot \mathbf{x}\| = c \cdot \|\mathbf{x}\|$

Relevance Ranking

- A query is represented as a document vector $\mathbf{q}$
- Compute similarity of $\mathbf{q}$ with all retrieved document vectors $\mathbf{d}$
  - similarity is computed as the cosine of the angle between the query vector and the document vector

$$\cos(\theta) = \frac{\mathbf{q} \cdot \mathbf{d}}{\|\mathbf{q}\| \cdot \|\mathbf{d}\|} = \frac{\sum_{i=1}^{n} d_{\mathbf{q},i} \, d_{\mathbf{d},i}}{\sqrt{\sum_{i=1}^{n} d_{\mathbf{d},i}^2} \cdot \sqrt{\sum_{i=1}^{n} d_{\mathbf{q},i}^2}}$$

- Rank the documents highest that have the smallest angle with the query
- Problem:
  - Web queries are too short
  - typically no good weights for query terms available

> will be covered later

Evaluation of a Retrieval Result

all documents

relevant documents

retrieved & relevant

retrieved documents
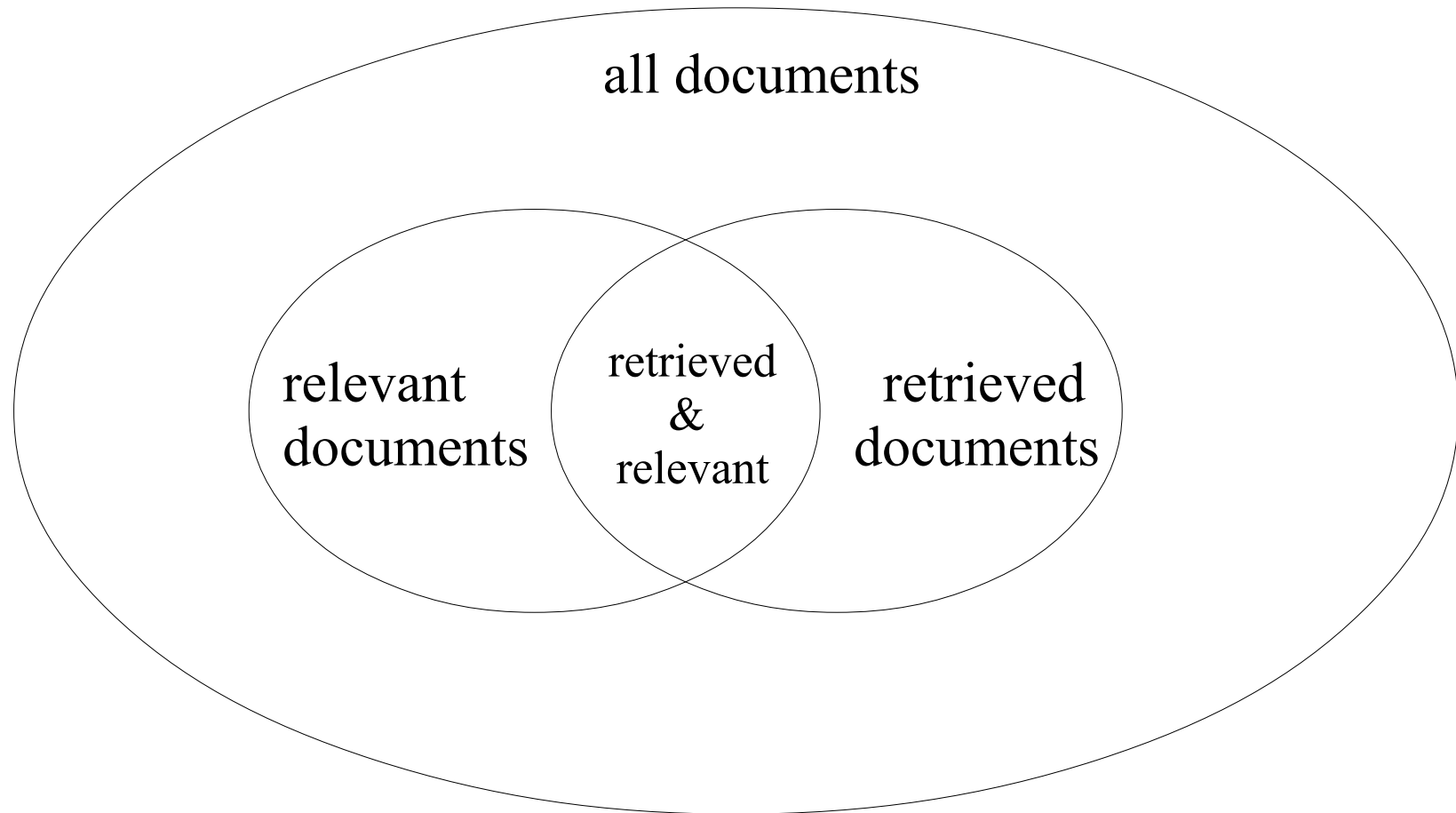
Evaluation - Accuracy

Confusion Matrix:

	retrieved	not retrieved	
Is relevant	a	c	$a+c=\lvert D_q\rvert$
Is not relevant	b	d	$b+d=\lvert D\setminus D_q\rvert$
	$a+b$	$c+d$	$\lvert D\rvert$

- **Accuracy:** percentage of correctly retrieved documents

$$accuracy = \frac{a+d}{\lvert D\rvert}$$

Recall and Precision

- Accuracy is not a good evaluation for IR
  - Accuracy can be made arbitrarily high by adding irrelevant documents to the document base (increasing d)
  - Accuracy must be interpreted relative to *default accuracy* (accuracy of the learner that always predicts majority class)
- Alternative:

  - **Recall**: Percentage of retrieved relevant documents among all relevant documents

  - **Precision**: Percentage of retrieved relevant documents among all retrieved documents

$$R = \frac{a}{a+c}$$

$$P = \frac{a}{a+b}$$

F-Measure

- Weighted harmonic mean of recall and precision

$$F_\beta = \frac{(\beta^2 + 1) \cdot P \cdot R}{\beta^2 \cdot P + R}$$

  - equivalent form for $\alpha = \dfrac{\beta^2}{\beta^2 + 1}$: $F_\alpha = \dfrac{1}{\alpha \dfrac{1}{R} + (1 - \alpha) \dfrac{1}{P}}$

- The parameter β can be used to trade off the relative importance of recall and precision

  - $F_0 = P$
  - $F_\infty = R$

  - F_1: P and R equally weighted

  - F_2: recall is four times more important than precision

  - $F_{0.5}$: precision is four times more important than recall

Recall-Precision Tradeoff

- Recall and Precision form a trade-off:

  - Precision can typically be increased by decreasing recall

  - Recall can typically be increased by sacrificing precision

  - e.g.: 100% recall can always be obtained by retrieving all documents
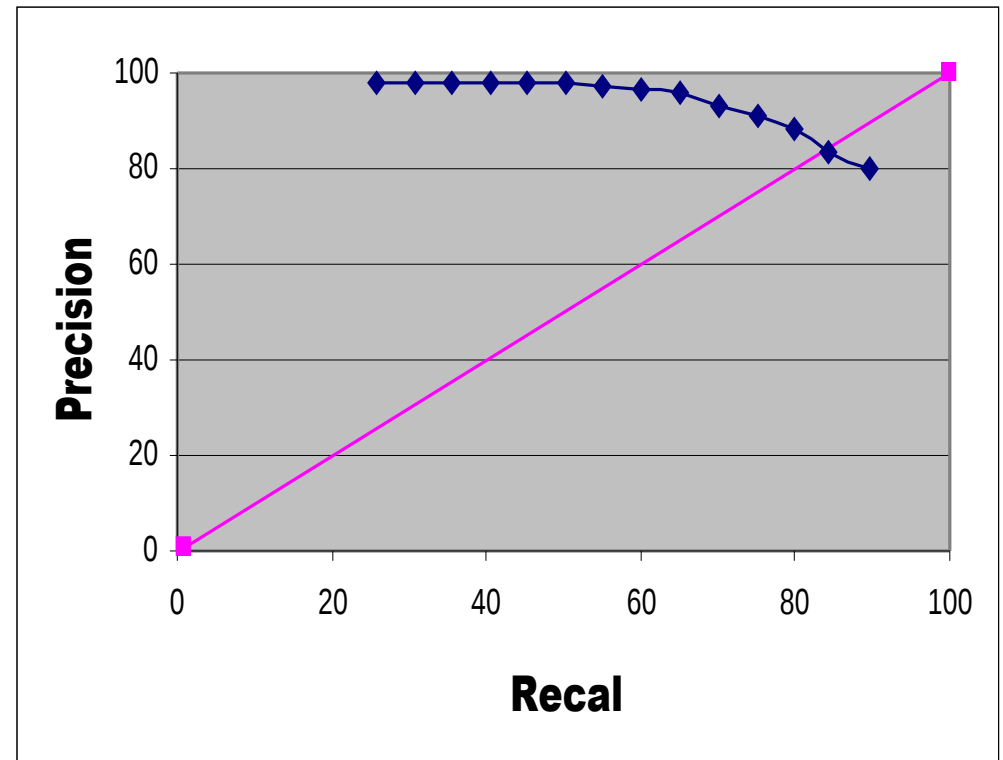
- Recall/Precision Curves

  - Trade-off can be visualized by plotting precision values over the corresponding recall values

Evaluating Performance in Practice

- Given benchmark
  - Corpus of documents D
  - A set of queries Q
  - For each query $\mathbf{q} \in Q$ an exhaustive set of relevant documents $D_q \subseteq D$ identified manually
- Each query is submitted to the system
  - result is a ranked list of documents $(\mathbf{d}_1, \mathbf{d}_2, \ldots, \mathbf{d}_n)$
  - compute a 0/1 relevance list $(r_1, r_2, \ldots, r_n)$

$$r_i = \begin{cases} 1 & \text{if } \mathbf{d}_i \in D_q \\ 0 & \text{otherwise} \end{cases}$$

- recall $R = \dfrac{1}{|D_q|} \sum_{1 \le i \le n} r_i$, precision $P = \dfrac{1}{n} \sum_{1 \le i \le n} r_i$

k	r_k
1	1
2	0
3	1
4	1
5	0
6	1
7	0
8	0
9	1
10	0
11	0
12	0
13	0
14	0
15	1
16	0
17	0
18	0
19	0
20	0

Chakrabarti and Ramakrishnan
J. Fürnkranz

Recall and Precision at Rank

- In many cases, we are not only interested in recall and precision of all retrieved documents
  - but only of the top k documents (those that we can browse)

- Recall at rank
  - Fraction of all relevant documents included in $\left(\mathbf{d}_1, \mathbf{d}_2, \ldots, \mathbf{d}_k \right)$

  - $R[k] = \dfrac{1}{|D_q|} \displaystyle\sum_{1 \le i \le k} r_i$

- Precision at rank
  - Fraction of the top $k \ge 1$ responses that are actually relevant.

  - $P[k] = \dfrac{1}{k} \displaystyle\sum_{1 \le i \le k} r_i$

Recall and Precision at Rank

k	r_k
1	1
2	0
3	1
4	1
5	0
6	1
7	0
8	0
9	1
10	0
11	0
12	0
13	0
14	0
15	1
16	0
17	0
18	0
19	0
20	0
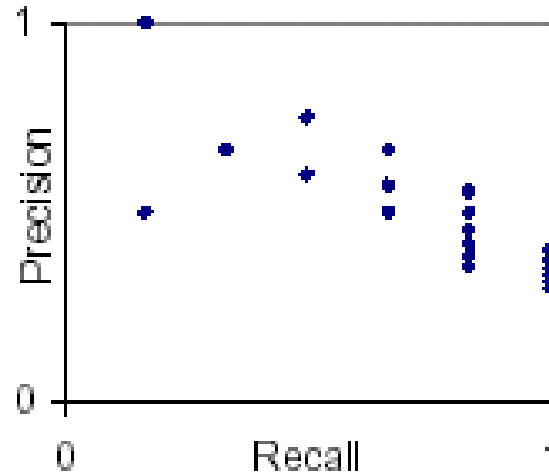
Precision at rank plotted against recall at rank for the given relevance vector.

Interpolated Precision

- Goal:
  - get a precision value for *each* recall point

- Simple strategy:
  - take the maximum precision obtained for the query for any recall greater than or equal to the current recall value r
  - basic idea: the best achievable precision for a given recall is shown

- can be used for combining results from multiple queries
  - e.g., to evaluate the performance of a search engine over multiple queries
  - average the interpolated precision values for each of a set of fixed recall levels and plot the result for this recall level

Interpolated Precision

k	r_k
1	1
2	0
3	1
4	1
5	0
6	1
7	0
8	0
9	1
10	0
11	0
12	0
13	0
14	0
15	1
16	0
17	0
18	0
19	0
20	0
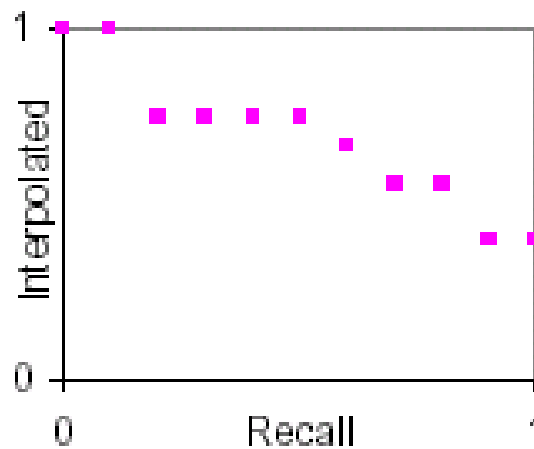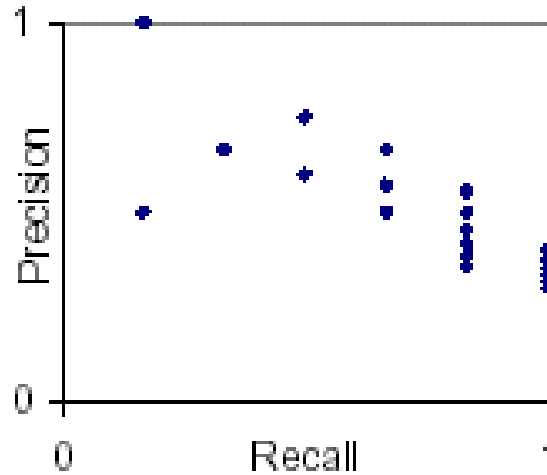
Precision and interpolated precision plotted against recall for the given relevance vector.

r_k

Sample Curve Before and After Interpolation
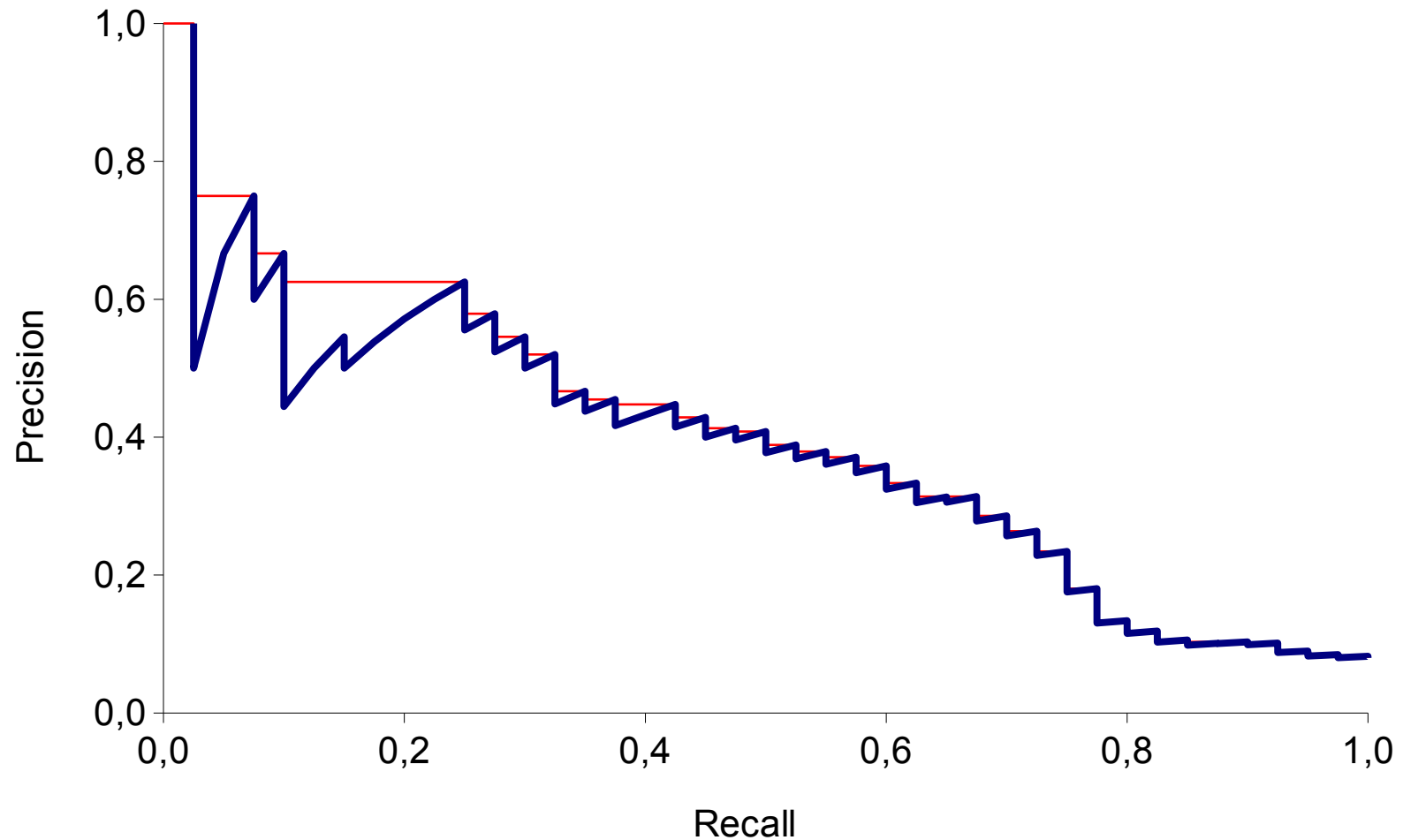
Summary Measures

Summary measures for evaluating the shape of the R/P curve:

- Average Precision
  - the average of all precision values at rank positions with relevant documents

  $$\mathrm{avg}\,P = \frac{1}{|D_q|} \sum_{1 \leq k \leq |D|} r_k \cdot P[k]$$

  - $\mathrm{avg}P = 1$ iff
    - engine retrieves all relevant documents and
    - ranks them ahead of any irrelevant document

- 11point Average Precision
  - average the 11 interpolated precision values for fixed recall levels of $0, 0.1, 0.2, ... 0.9, 1.0$

Summary Measures

k	r_k
1	1
2	0
3	1
4	1
5	0
6	1
7	0
8	0
9	1
10	0
11	0
12	0
13	0
14	0
15	1
16	0
17	0
18	0
19	0
20	0
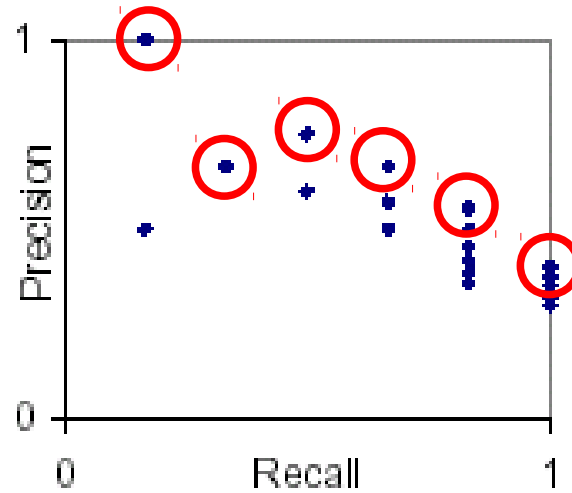
Average Precision
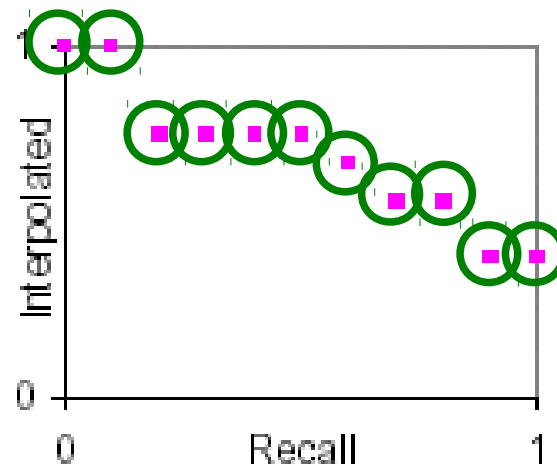
11pt Average Precision

Breakeven Point

- Another simple summary measure
  - the point where recall equals precision
  - Estimated by linear interpolation
- Assumption:
  - Distance to origin determines quality of recall/precision curve
- Example:

Precision	Recall
72.38	97.88
75.09	97.76
80.01	97.18
85.02	96.20
90.00	**93.89**
94.41	**88.57**
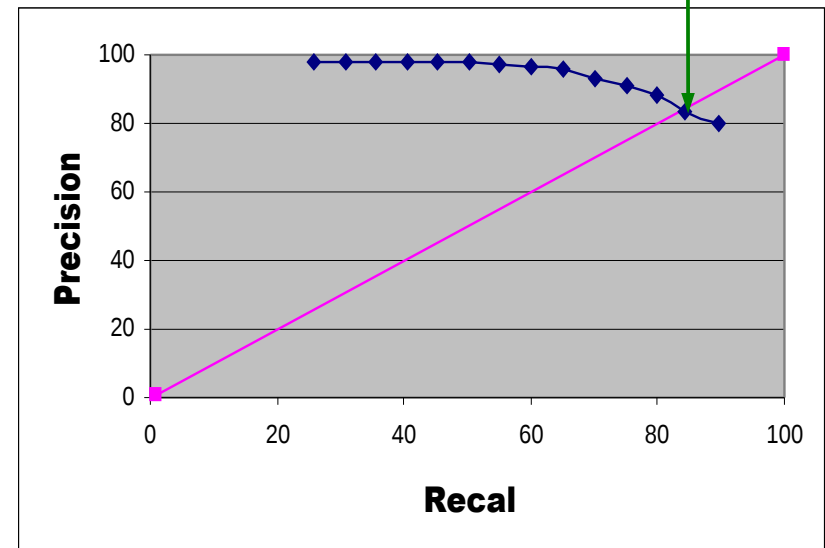
$$B = \frac{R_2 \cdot P_1 - R_1 \cdot P_2}{R_2 - R_1 + P_1 - P_2} = 91,76$$

J. Fürnkranz

Discounted Cumulative Gain

- Discounted Cumulative Gain (DCG)
  - Key idea:
    - average precision gives equal weight to all positions
    - but top positions in the ranking are more important and should receive higher weights!
  - Approach:
    - Discount the relevance factor r_i with the logarithm of i

$$DCG[k] = \sum_{1 \leq i \leq k} \frac{r_i}{\log_2(i+1)}$$

- Normalized Discounted Cumulative Gain (NDCG)
  - Normalize the DCG-value of the ranking with the optimal DCG-value for this query (i.e., the DCG value for the perfect ranking)

$$NDCG[k] = \frac{DCG[k]}{ODCG[k]}$$

DCG Example

- DCG at ranking position 5:

$$DCG[5] = \frac{1}{\log_2(2)} + \frac{0}{\log_2(3)} + \frac{1}{\log_2(4)} + \frac{1}{\log_2(5)} + \frac{0}{\log_2(6)}$$

$$= 1 + 0 + \frac{1}{2} + \frac{1}{2.34} + 0 = \mathbf{1.93}$$

- NDCG at ranking position 5:
  - we have 6 relevant documents, the perfect has 5 relevant docs at the first 5 places

$$ODCG[5] = \frac{1}{\log_2(2)} + \frac{1}{\log_2(3)} + \frac{1}{\log_2(4)} + \frac{1}{\log_2(5)} + \frac{1}{\log_2(6)}$$

$$= 1 + \frac{1}{1.585} + \frac{1}{2} + \frac{1}{2.34} + \frac{1}{2.585} = \mathbf{2.95}$$

$$NDCG[5] = \frac{DCG(5)}{ODCG(5)} = \mathbf{0.655}$$

k	r_k
1	1
2	0
3	1
4	1
5	0
6	1
7	0
8	0
9	1
10	0
11	0
12	0
13	0
14	0
15	1
16	0
17	0
18	0
19	0
20	0

J. Fürnkranz

Multi-Level Relevance Scores

- user Feedback about relevance need not be binary
  - users may give feedback on multiple levels

- Example:
  - How relevant is this page on a scale from 0 to 5?

- DCG and NDCG can be directly generalized to this case
  - r_i can then have values 0 to 5 instead of 0 to 1
  - the optimal ranking for the computation of NDCG is any ranking that sorts all pages with a higher score before all pages with a lower score

Improving Retrieval Efficiency

- Relevance Feedback
    - user is willing to provide feedback
- Clustering Search Results
    - results are summarized into different groups
- Meta-Search Engines
    - query multiple engines and combine results
- Hyperlink-based Ranking
    - later in this course

J. Fürnkranz

Relevance Feedback

1. Present an result for query $\mathbf{q}_i$

2. Get feedback from the user
   - *Explicit Relevance Feedback:*
     User marks documents as relevant or not
   - *Implicit Relevance Feedback:*
     User's actions are observed
     (e.g., does he view the document or not?)

3. Formulate new query $\mathbf{q}_{i+1}$ by enriching original query with query terms from relevant documents
   - e.g., select by log odds ratio, add relevant document vectors, subtract irrelevant document vectors

4. $i = i + 1$, Goto 1.

PROBLEM: increased effort for the user, only feasible for long-term monitoring of interactions (e.g., WebWatcher)

Relevance Feedback: Example

- Image search engine
  http://nayana.ece.ucsb.edu/imsearch/imsearch.html
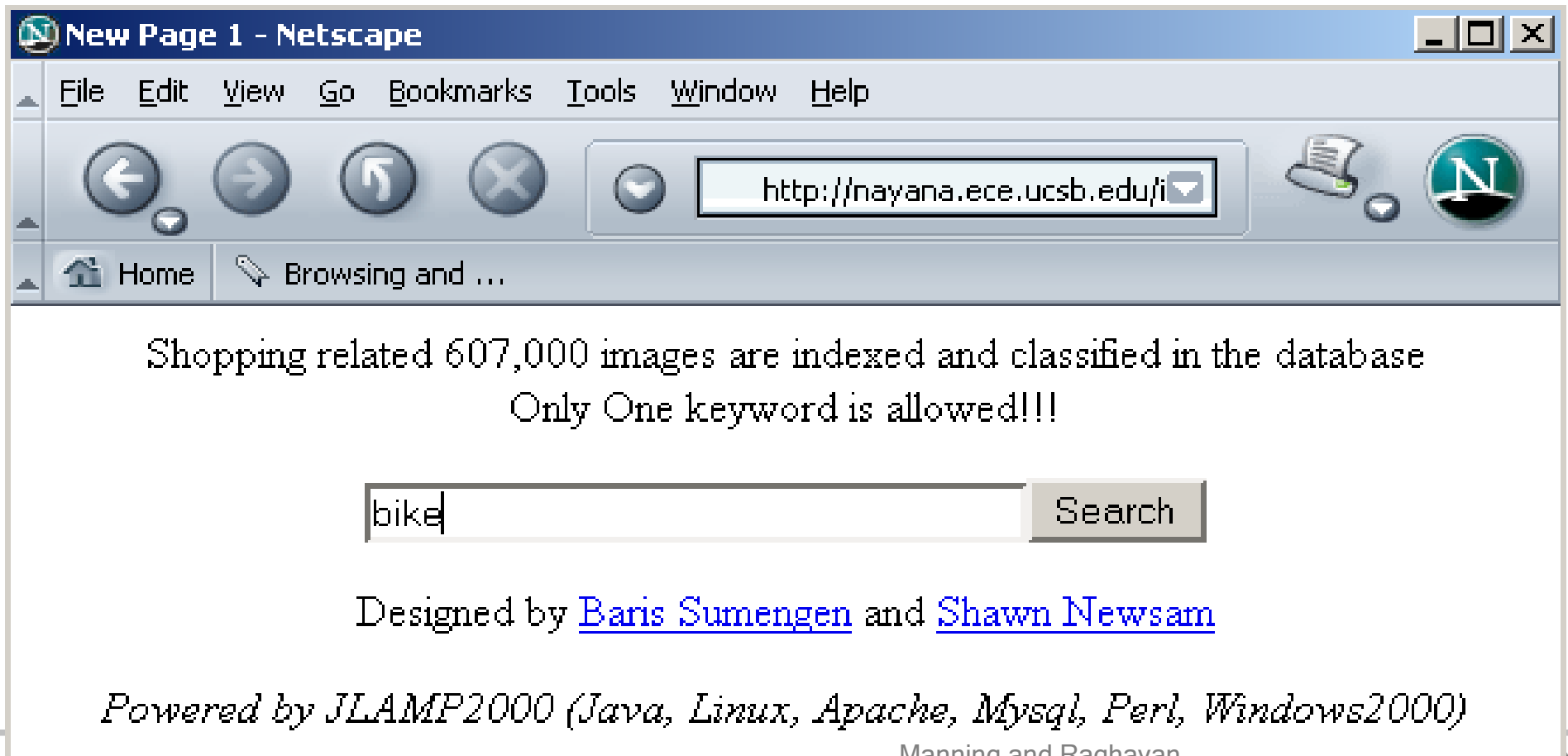
Results for Initial Query

Manning and Raghavan   J. Fürnkranz

Relevance Feedback

Manning and Raghavan   J. Fürnkranz
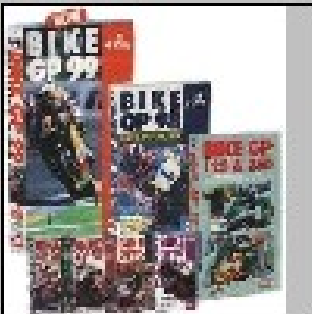
Results after Relevance Feedback

Manning and Raghavan   J. Fürnkranz
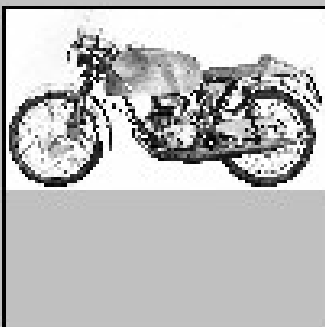
Rocchio Algorithm

- increase weight of terms that appear in relevant documents $\mathbf{r}_j \in R$

- decrease weight of terms that appear in irrelevant documents $\mathbf{i}_j \in I$

$$\mathbf{q}_{i+1} = \alpha \cdot \mathbf{q}_i + \beta \cdot \sum_j \mathbf{r}_j - \gamma \cdot \sum_j \mathbf{i}_j$$

- typical parameter settings $\alpha = 1 \, ; \, \beta = \dfrac{1}{|R|} \, ; \, \gamma = \dfrac{1}{|I|}$

- A few iterations of this can significantly improve performance

Pseudo Relevance Feedback

- Pseudo-relevance feedback
  - R and I generated automatically
    - E.g.: Cornell SMART system
    - top 10 documents reported by the first round of query execution are included in R
  - γ typically set to 0; I not used

- Not a commonly available feature
  - Web users want instant gratification
  - System complexity
    - Executing the second round query slower and expensive for major search engines

Clustering of Search Results

- Search results are often ambigous
  - e.g., 'jaguar' returns documents on cars and documents on animals
  - user is typically only interested in one meaning

- Solution: Clustering algorithms
  - detect groups of pages that have similar outcomes
  - basic idea:
    - sort objects into classes in order to maximize
      - intra-class similarity
      - inter-class dissimilarity

J. Fürnkranz

Clusty

jaguar

Search

clusters   sources   sites

All Results (253)   remix

⊖ **Pictures** (37)

⊖ **Parts** (29)

　● **Jaguar Parts And Accessories** (4)

　● **Specialize** (4)

　● **Jaguar car** (5)

　● **New And Used Jaguar Parts** (3)

　● **Dealer, Sales** (3)

⊕ **Cars, Parts** (4)

　● **Jaguar Wheels** (3)

　● **Jaguar X-Type** (2)

　● **Other Topics** (5)

⊕ **Club** (25)

⊖ **Panthera onca** (15)

　⊕ **Largest cat** (6)

　● **Species** (3)

　● **Mammal Of The Felidae Family And One Of Four "Big Cats"** (2)

　● **Facts, And Pictures About Jaguar** (2)

　● **Other Topics** (3)

⊕ **Jacksonville** (11)

⊕ **Reviews** (15)

⊕ **Land Rover** (12)

⊕ **Animal** (8)

⊕ **Cars for sale** (9)

Top **251** results of at least **73,900,000** retrieved for the query **jaguar** (definition) (details)

Jaguar Fahrzeuge - Finden Sie Ihr Traumauto inklusive Finanzierung aus über 470.000 PKW! - www.gebrauchtwagen.de

Ask a Jaguar Mechanic Now - 12 **Jaguar** Mechanics Are Online! Ask a Question, Get an Answer ASAP. - Jaguar.JustAnswer.com

http://www.clusty.com

1. **Jaguar - Wikipedia, the free encyclopedia**
The **jaguar** (Panthera onca) is a big cat, a feline in the Panthera genus, and is the only Panthera species found in the Americas. The **jaguar** is the third-largest feline after the tiger ... en.wikipedia.org/wiki/**Jaguar** · Wikipedia on Bing
en.wikipedia.org/wiki/Jaguar - [cache] - Bing, Ask, Yahoo!

2. **Jaguar USA - Jaguar Cars**
XF SPECIAL OFFER. Learn about an exclusive lease offer on the 2010 **Jaguar** XF > LOCATE A DEALER . Search for your nearest **Jaguar** Dealer now > STAY INFORMED www.**jaguar**.com/us/en/
www.jaguar.com/us/en - [cache] - Bing, Yahoo!, Ask

3. **Jaguar International - Jaguar International**
Our mission at **Jaguar** has been to create and build beautiful fast cars. The XK, XF, X-TYPE and now All New XJ bring the exhilaration of driving to life. ... JAGUAR'S 75TH ANNIVERSARY
www.jaguar.com - [cache] - Ask, Open Directory, Yahoo!

4. **Jaguar Cars - Wikipedia, the free encyclopedia**
Jaguar Cars Ltd., better known simply as **Jaguar** is a British luxury car manufacturer, headquartered in Coventry, England. It has been a wholly-owned subsidiary of the Indian ... en.wikipedia.org/wiki/**Jaguar**_Cars · Wikipedia on Bing
en.wikipedia.org/wiki/Jaguar_Cars - [cache] - Bing, Yahoo!

5. **Jaguar : Mysterious Cat of the Amazon**
Compares **jaguars** and leopards and provides information about the animal's shrinking habitat and relationship with man.
www.bluelion.org/jaguar.htm - [cache] - Yahoo!, Ask

6. **The Official Website of the Jacksonville Jaguars ...**
Jaguars TicketExchange: Buy and sell tickets; Fanzone. Message Board; Jaxson de Ville; Downloads; D-Line; Champions Club; Scoreboard Messages; Contests; Search and Win! jaguars.com/
jaguars.com - [cache] - Bing, Open Directory

7. **Jaguar - Cars.com**
New and used **Jaguar** cars and trucks. See the latest **Jaguar** models and find a **Jaguar** for sale in your area. www.cars.com/jaguar · Cached page
www.cars.com/jaguar - [cache] - Bing, Yahoo!

Meta-search systems

- Take the search engine to the document
  - Forward queries to many geographically distributed repositories
    - Each has its own search service
  - Consolidate their responses.
- Advantages
  - Automatically perform non-trivial query rewriting
    - Suit a single user query to many search engines with different query syntax
  - Surprisingly small overlap between crawls
- Consolidating responses
  - Function goes beyond just eliminating duplicates
  - Search services do not provide standard ranks which can be combined meaningfully

Example: MetaCrawler

http://www.metacrawler.com/

Web Search Engines

- Crawler
- Indexer
- Query Interface
- Ranker
➔ Scalability
  - Index Sizes
  - Estimating the Size of the Web
  - Coverage

Millions of Web Pages Indexed — 2001

GG	FAST	AV	INK	NL
2000 / 1500	625	550	500 / 110	390
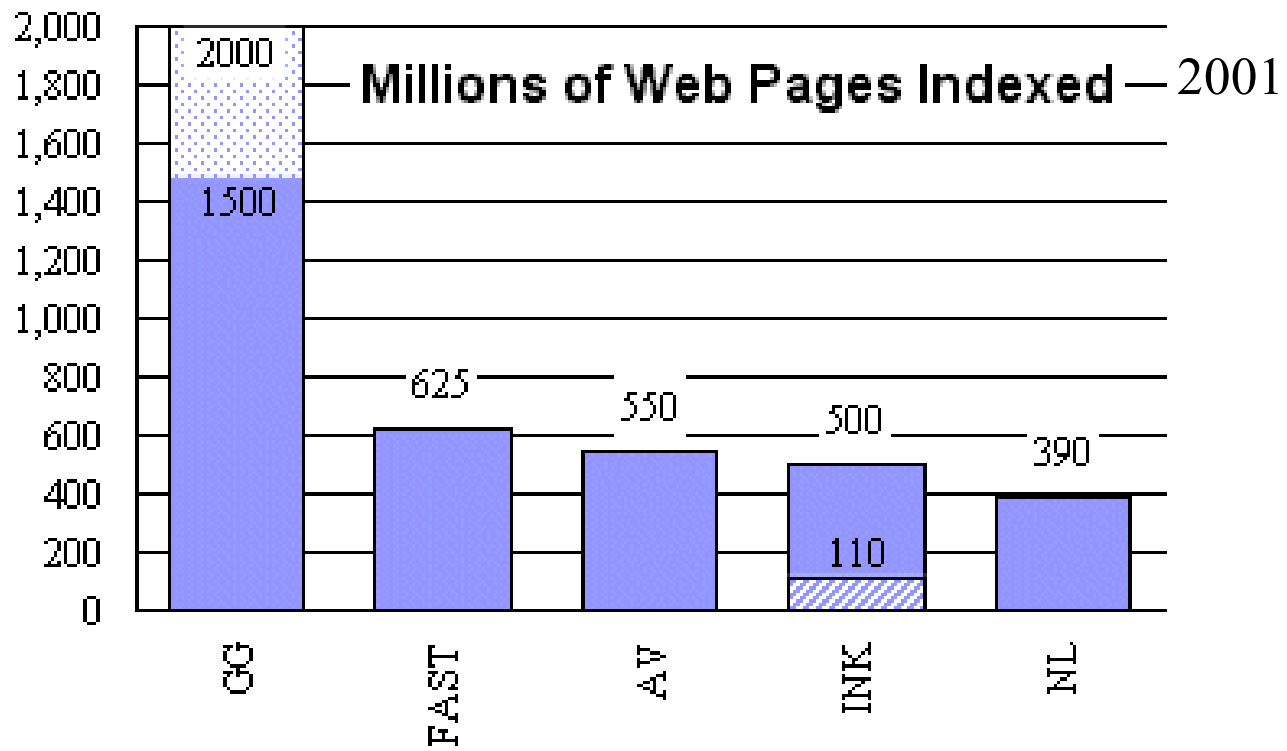
Search Engine Sizes

Source:
searchenginewatch.com

Search Engine Sizes
(millions of web pages)

GG — ATW — INK — TMA — AV

Searches per Day

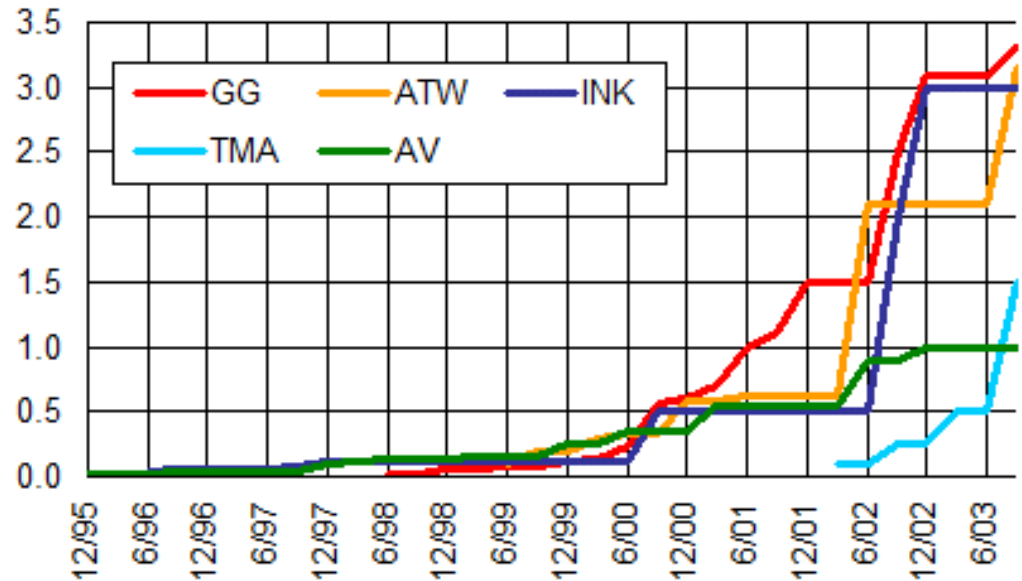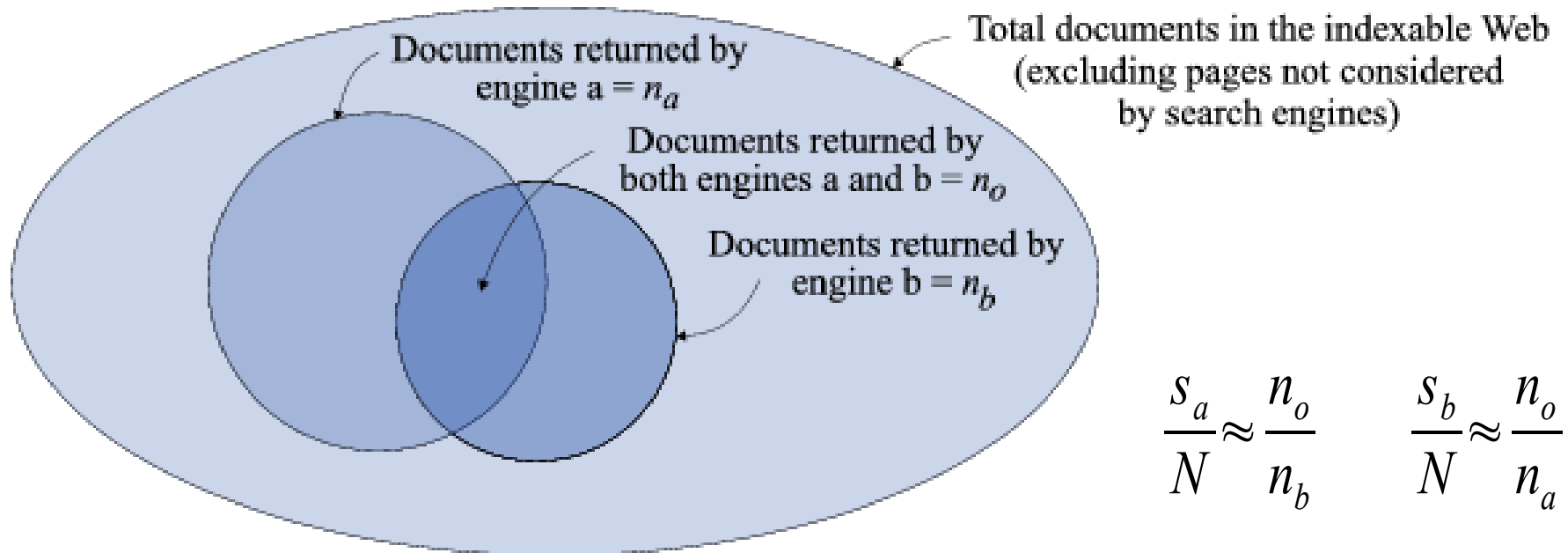| Service	Searches Per Day	As Of/Notes
Google	250 million	February 2003 (as reported to me by Google, for queries at both Google sites and its partners)
Overture	167 million	February 2003 (from Piper Jaffray's The Silk Road for Feb. 25, 2003 and covers searches at Overture's site and those coming from its partners.)
Inktomi	80 million	February 2003 (Inktomi no longer releases public statements on searches but advised me the figure cited is "not inaccurate")
LookSmart	45 million	February 2003 (as reported in interview and includes searches with all LookSmart partners, such as MSN)
FindWhat	33 million	January 2003 (from interview and covers searches at FindWhat and its partners)
Ask Jeeves	20 million	February 2003 (as reported to me by Ask Jeeves, for queries on Ask, Ask UK, Teoma and via partners)
AltaVista	18 million	February 2003 (from Overture press conference on purchase of FAST's web search division)
FAST	12 million	February 2003 (from Overture press conference on purchase of FAST's web search division)

Source
searchenginewatch.com

Search Engine Market Share

Worldwide Search Market Overview, July 2009 vs. July 2008				
	Searches (Millions)			
	July 2008	July 2009	Change (%)	Share (%)
Total Internet	*80,554*	*113,685*	*41*	*100*
Google sites	48,666	76,684	58	67.5
Yahoo! Sites	8,689	8,898	2	7.8
Baidu.com Inc.	7,413	7,976	8	7
Microsoft Sites	2,349	3,317	41	2.9
eBay	1,223	1,723	41	1.5
NHN Corporation	1,243	1,526	23	1.3
Ask Network	929	1,291	39	1.1
Yandex	663	1,290	94	1.1
AOL LLC	1,148	1,023	-11	0.9
Facebook.com	743	879	18	0.7

Notes:
Audience includes Internet users, ages 15 and older, at home and work. It excludes Internet activity from public computers, such as Internet cafes, and access from mobile phones or PDAs.

Source: comScore qSearch, 2009

Estimating the Size of the Web

- Lawrence & Giles, *Science* 1998

- Procedure

  - Submitted 575 queries from real users to several search engines

  - Tried to avoid difficulties originating from different indexing and retrieval schemes of the search engines

  - Obtained different size estimates for number of indexed documents from the pairwise overlap of search engines

  - The largest was 320,000,000 pages

- Assumption

  - pages indexed by search engines are independent

  - unrealistic, hence true estimate is larger

Estimating the Size of the Web (2)

Documents returned by engine a = n_a

Documents returned by both engines a and b = n_o

Documents returned by engine b = n_b

Total documents in the indexable Web (excluding pages not considered by search engines)

$$\frac{s_a}{N} \approx \frac{n_o}{n_b} \qquad \frac{s_b}{N} \approx \frac{n_o}{n_a}$$

s_a = total number of pages indexed by search engine a
s_b = total number of pages indexed by search engine b

$$N \approx s_a \frac{n_b}{n_o} \approx s_b \frac{n_a}{n_o}$$

Graphic from NEC Research, http://www.neci.nj.nec.com/~lawrence/websize.html

J. Fürnkranz

Search Engine Coverage
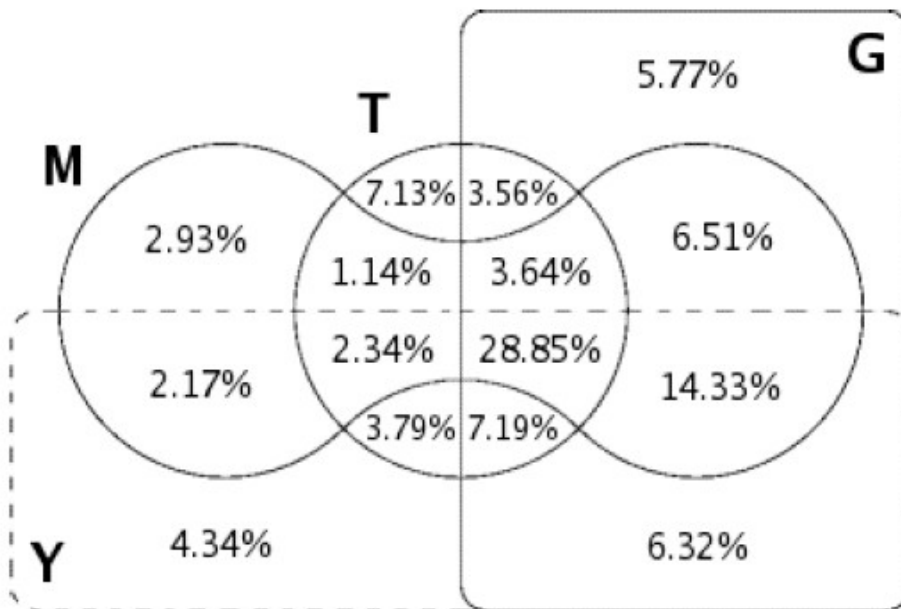
- Based on the estimated size of the Web 1998
- The best engine indexes only 32% of the "indexable Web"

Graphic from NEC Research, http://www.neci.nj.nec.com/~lawrence/websize.html

J. Fürnkranz
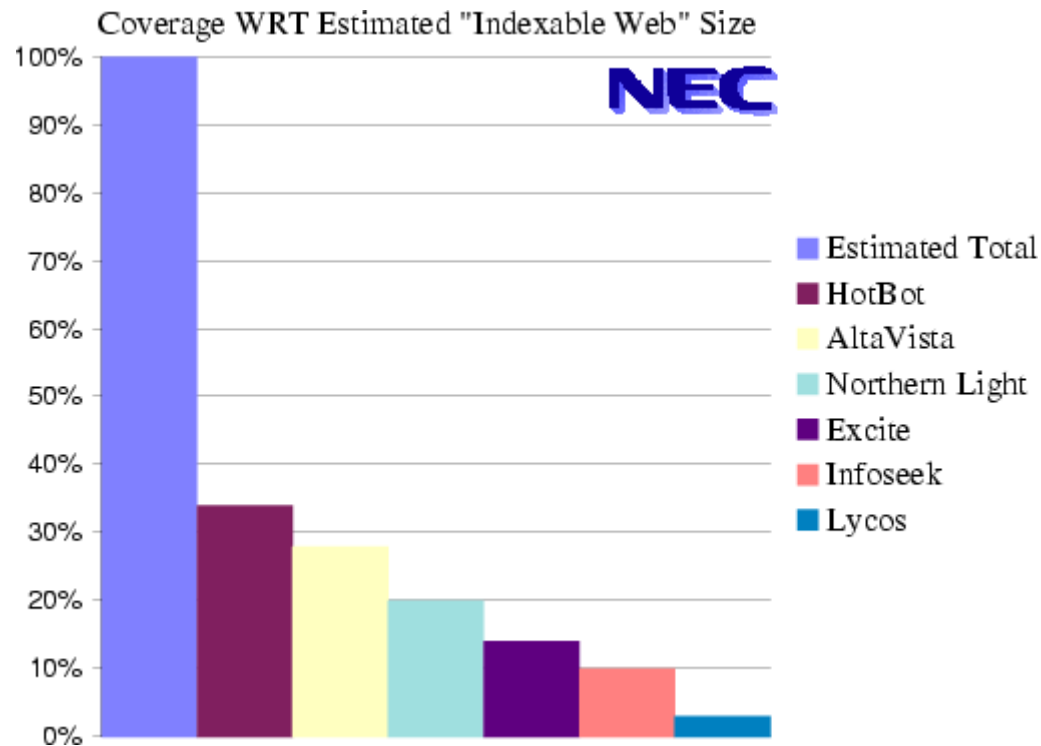
Newer Results

- Gulli & Signorini (WWW-14, 2005)
  - based on a similar study by Bharat & Broder (1998)
  - size of indexable web = 11.5 billion pages

Engines Coverage %

	Google	Msn	Teoma	Yahoo!
Coverage	76.27	61.87	57.70	69.37

Engines Intersections %

	Google	Msn	Teoma	Yahoo!
Google	-	55.23	35.96	56.04
Msn	78.42	-	49.87	67.30
Teoma	58.20	42.68	-	54.13
Yahoo!	68.45	49.56	44.98	-

- Google vs. Yahoo controversy (Cheney & Perry 2005)
  - as a result, Google stopped announcing index sizes

Search Engine Resources

- Search Engine Watch
  - http://searchenginewatch.com/resources/