

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

- Crawler <http://www.searchenginewatch.com>
 - 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

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.

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.: AltaVista, Northern Lights, 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

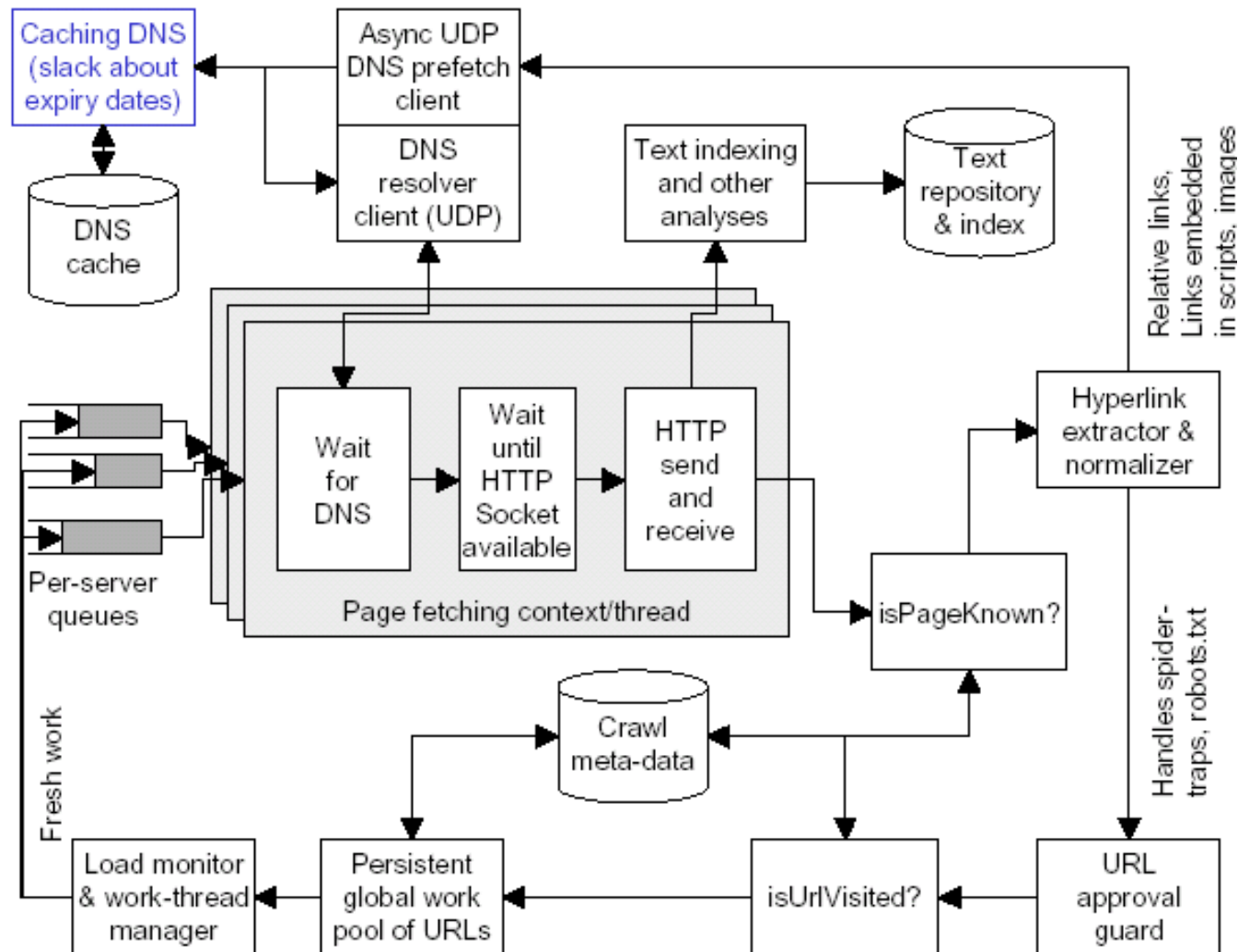
Crawling overheads

- 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 its 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.

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
 - specifies 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/>

Spider Traps: Solutions

- No automatic technique can be foolproof
- Check for URL length
- 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

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
 - Mirrored Web pages and sites
- 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 $(\text{hash}(u_1); v)$ and $(\text{hash}(u_2); v)$
- 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 successive 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

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

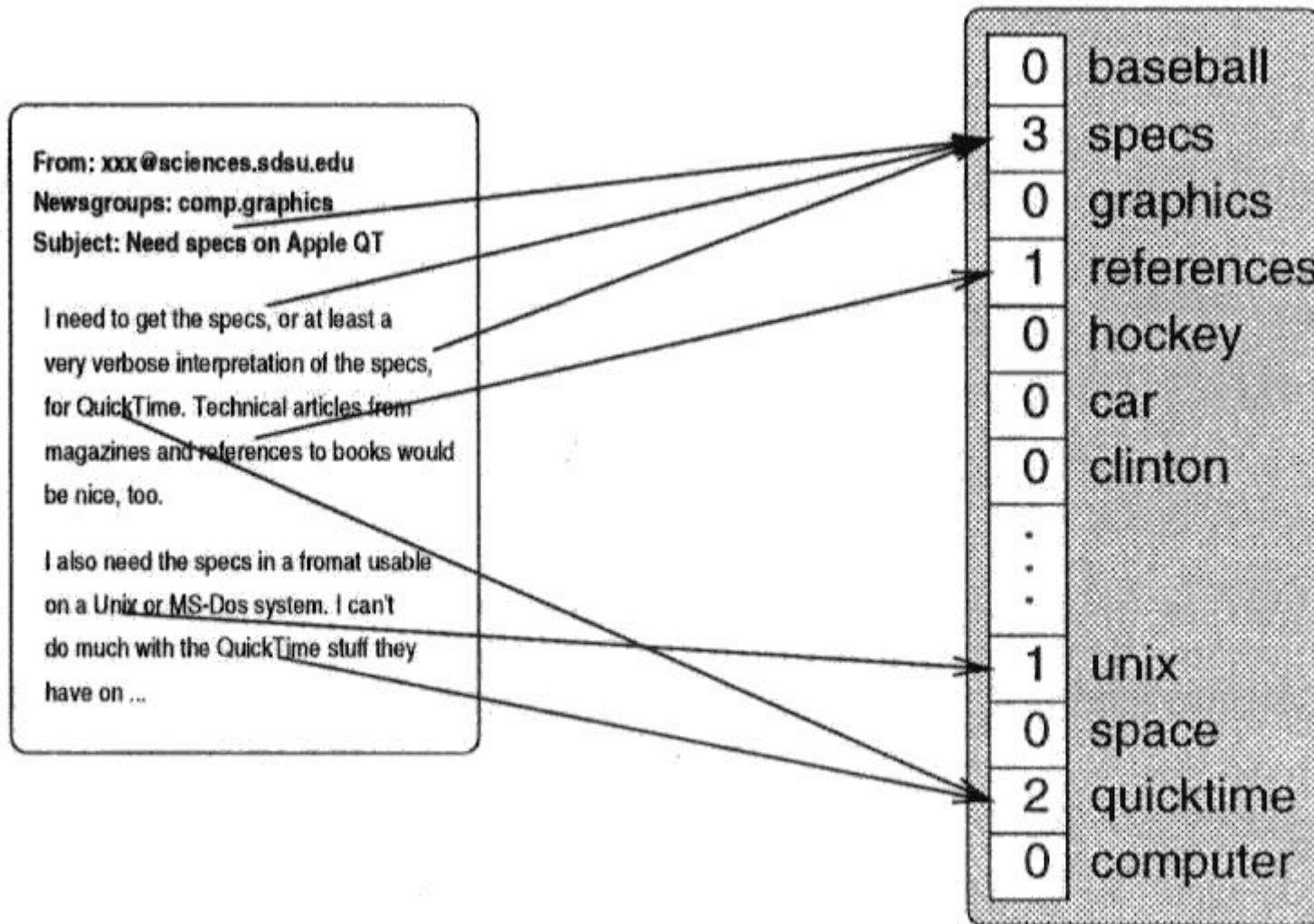
Web Search Engines

- 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

Document preprocessing: Tokenization

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

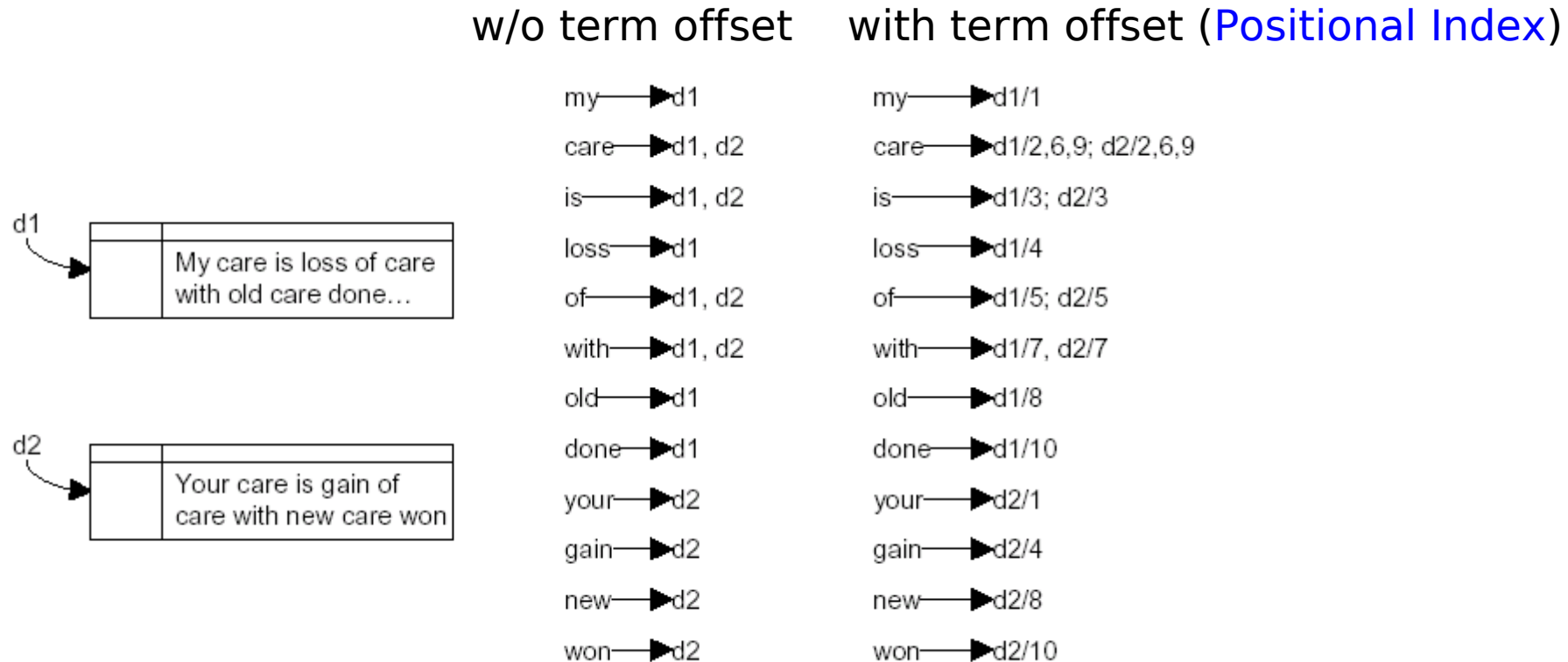
	D1	D2	D3
baseball	0	1	0
specs	3	2	0
graphics	0	0	2
.....
quicktime	2	0	1
computer	0	0	5

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.



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
 - Index size depends on average document size
- Rules of Thumb
 - A positional index is 2–4 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

Index Size Reduction by Filtering

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

Index compression techniques

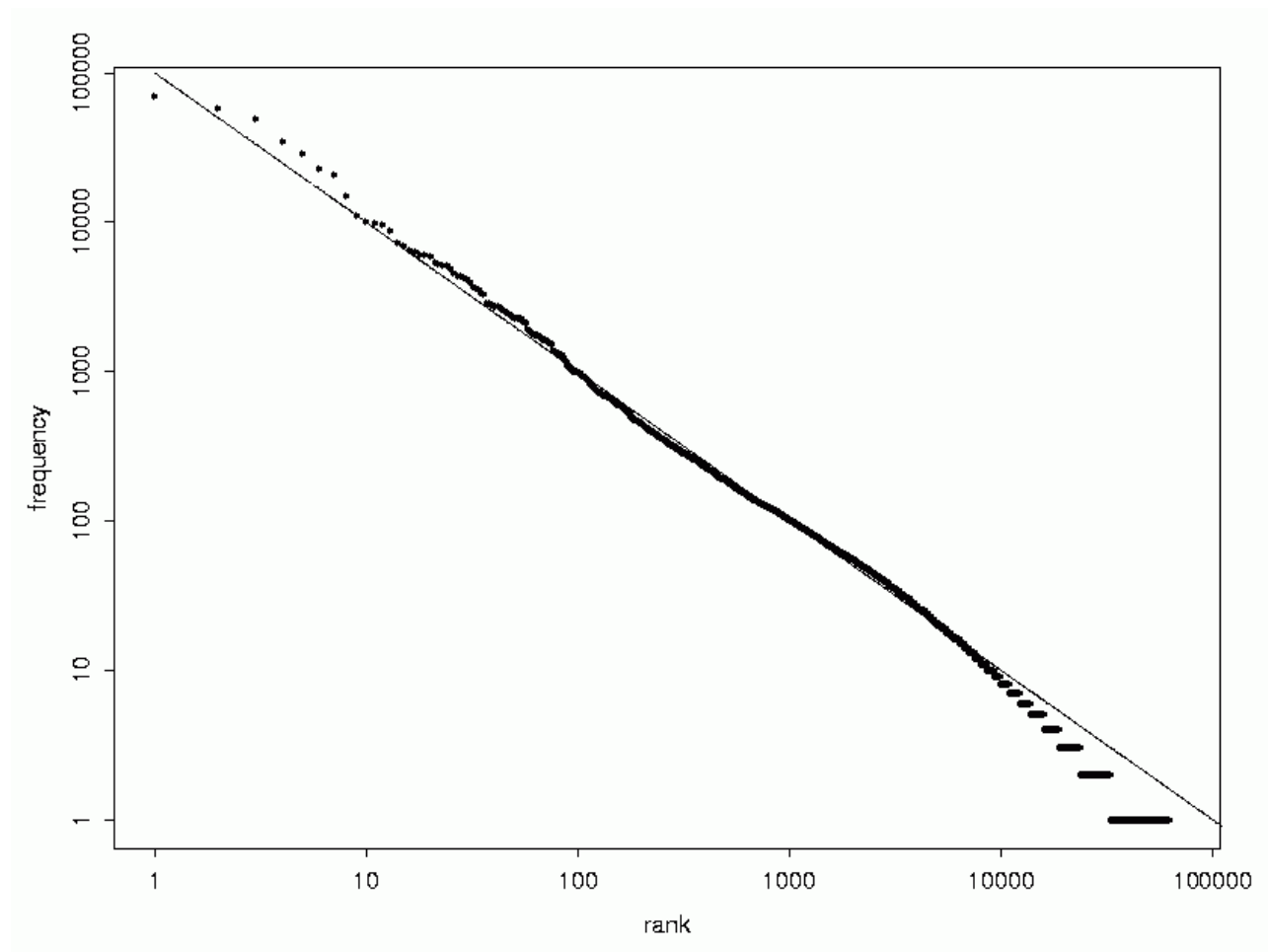
- Compressing the index so that much of it can be held in memory
 - Required for high-performance IR installations (as with Web search engines),
- Redundancy in index storage
 - Storage of document IDs.

Delta encoding:

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

Zipf's Law

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



Encoding gaps

Small gap must cost far fewer bits than a document ID.

- Binary encoding
 - regular encoding for integers
 - Optimal when all symbols are equally likely
- Unary code
 - the number n is represented with n consecutive 1's
 - optimal if probability of gaps of size n decays exponentially
- Gamma code $(Pr(n) = 2^{-n})$
 - Represent gap x as
 - Unary code for $1 + \lceil \log x \rceil$ followed by
 - $x - 2^{\lceil \log x \rceil}$ represented in binary ($\lceil \log x \rceil$ bits)
- Golomb codes
 - Further enhancement

Lossy compression mechanisms

- 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*)

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.

Other issues (contd..)

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

Web Search Engines

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

Simple Boolean Queries

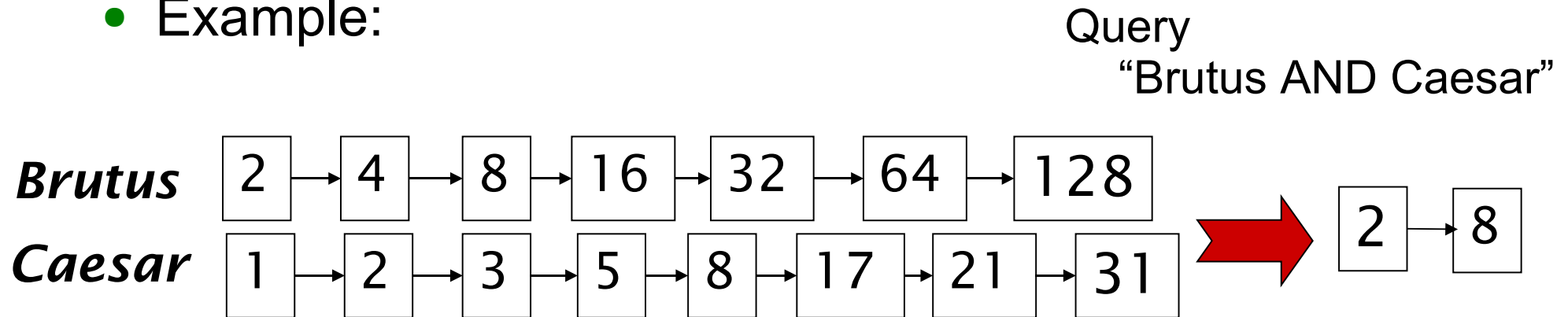
look up the inverted index

- **One-term queries** (T1):
⇒ look up term, return documents with non-zero entries
- **Conjunctive queries** (T1 AND T2):
⇒ Intersection of documents with non-zero entries
- **Disjunctive queries** (T1 OR T2):
⇒ Union of documents with non-zero entries
- **Negation** (NOT T1):
⇒ 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:

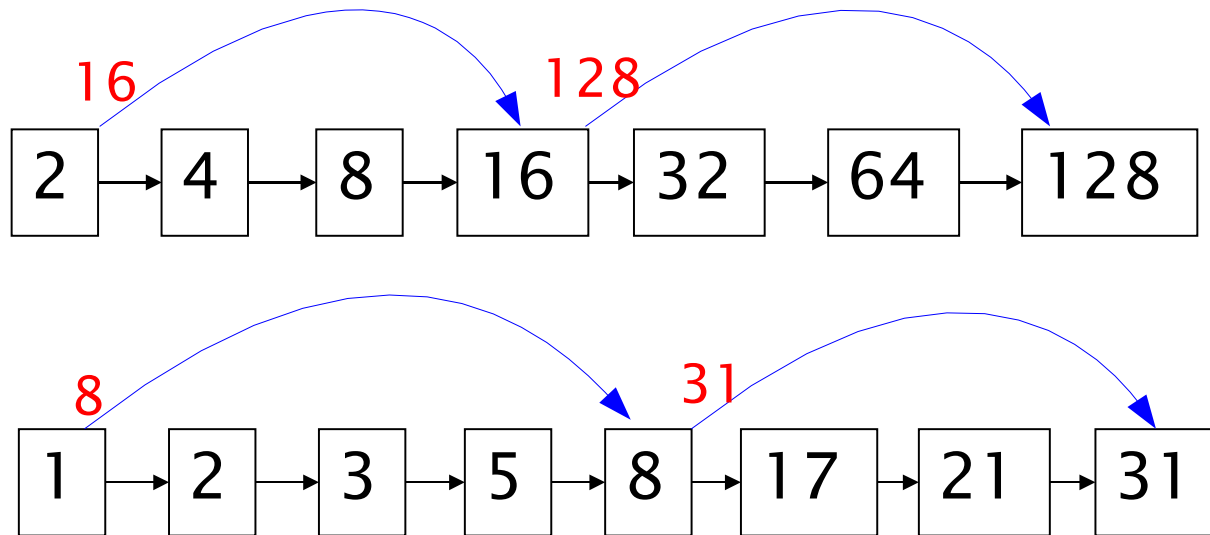


If the list lengths are m and n , the merge takes $O(m+n)$ operations.

Can we do better?

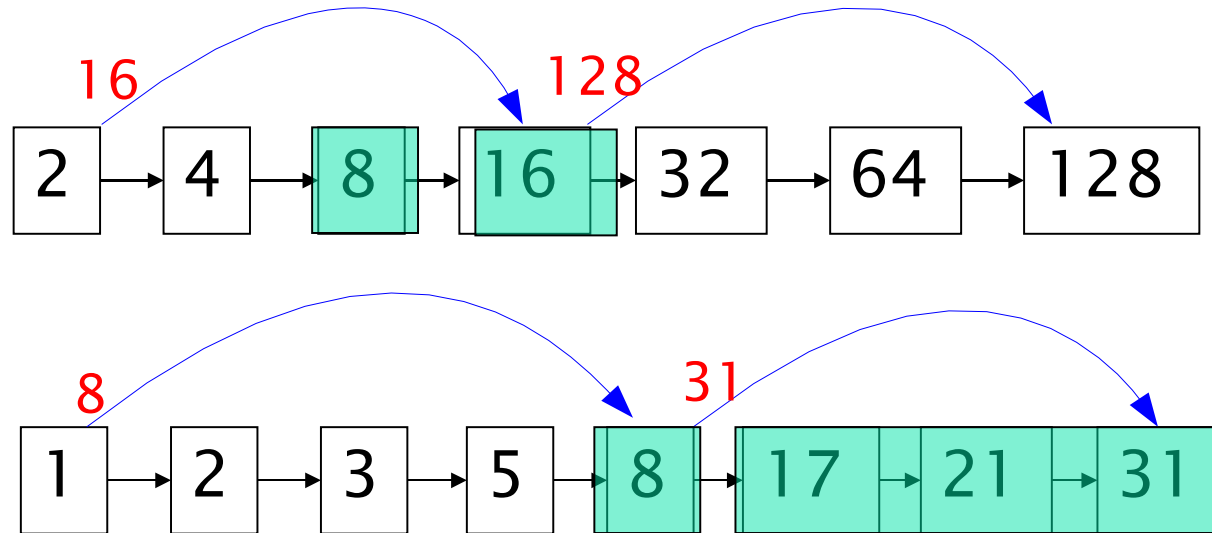
Yes, if index isn't changing too fast.

Augment Lists with Skip Pointers



- Why?
 - To skip postings that will not figure in the search results.
- 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.

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

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 Search Syntax



<http://www.altavista.com/help/search/syntax>

[Home](#) > [AltaVista Help](#) > [Search](#) > **Special search terms**

You can use these terms for both [basic](#) and [advanced](#) Web searches. For advanced searches, type these into the free-form Boolean box.

AND	Finds documents containing all of the specified words or phrases. Peanut AND butter finds documents with both the word peanut and the word butter.
OR	Finds documents containing at least one of the specified words or phrases. Peanut OR butter finds documents containing either peanut or butter. The found documents could contain both items, but not necessarily.
NOT	Excludes documents containing the specified word or phrase. Peanut NOT butter finds documents with peanut but not containing butter. NOT must be used with another operator, like AND. AltaVista does not accept *peanut NOT butter* ; instead, specify peanut NOT butter .
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.
url:text	Finds pages with a specific word or phrase in the URL. Use url:garden to find all pages on all servers that have the word <i>garden</i> anywhere in the host name, path, or filename.

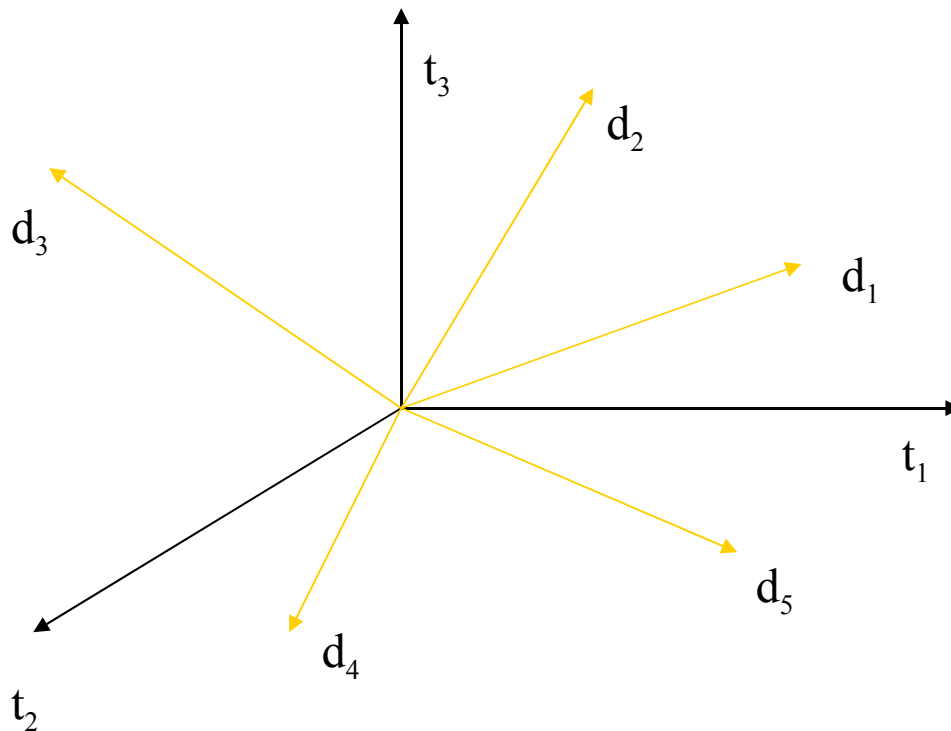
Web Search Engines

- Crawler
- Indexer
- Query Interface
- Ranker
 - The Vector-Space Model
 - Similarity-Based Ranking
 - Evaluation of Ranking Results
 - Recall and Precision
 - 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 the number of times the word occurs in the document (*term frequency - TF*)
 - a vector is a linear combination of the base vectors
 - linear algebra can be used for various computations

Intuition



Postulate: Documents that are “close together” in the vector space talk about the same things.

Term Weighting

- Boolean
 - possible values are only
 - 0 (term does not occur in document) or
 - 1 (term does occur)
- Term Frequency (TF)
 - term is weighted with the frequency of its occurrence in the text
- Term Frequency - Inverse Document Frequency (TF-IDF)
 - Idea: A term is characteristic for a document if
 - it occurs frequently in this document
 - occurs infrequently in other documents
 - divides TF by DF (or multiplies TF with IDF)
 - often used as an alternative term weighting approach

Term frequency

- Measures the frequency of the occurrence of a term in the document

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

- Other estimates are possible

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

- Cornell SMART system uses a smoothed version

$$\begin{aligned} TF(d, t) &= 0 && \text{if } n(d, t) = 0 \\ TF(d, t) &= 1 + \log(1 + n(d, t)) && \text{otherwise} \end{aligned}$$

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 $\frac{D}{|D_t|}$
 - in the SMART retrieval system
 - $IDF(t) = \log\left(\frac{1+|D|}{|D_t|}\right)$

Relevance ranking

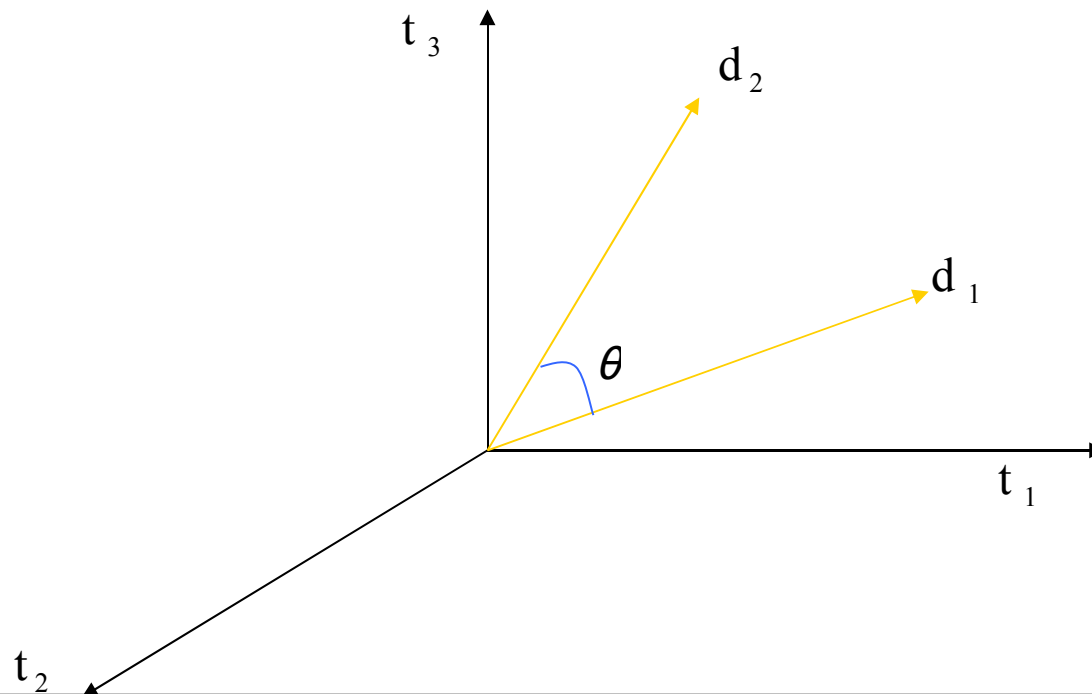
- Keyword queries
 - In natural language
 - Queries are not precise
 - entire set of matching documents for response unacceptable
 - Solution
 - Rate each document for how likely it is to satisfy the user's information need (relevance)
 - Sort in decreasing order of the score
 - Present results in a 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 d_1 and d_2 is the length (Euclidean Distance) of the vector $|d_1 - d_2|$.
- Why is this not a great idea?
 - We still haven't dealt with the issue of length normalization
 - Short documents would be more similar to each other by virtue of length, not topic
- However, we can implicitly normalize by looking at *angles* instead

Cosine similarity

- Distance between vectors d_1 and d_2 captured by the cosine of the angle x between them.
- Note – this is *similarity*, not distance
 - No triangle inequality for similarity.



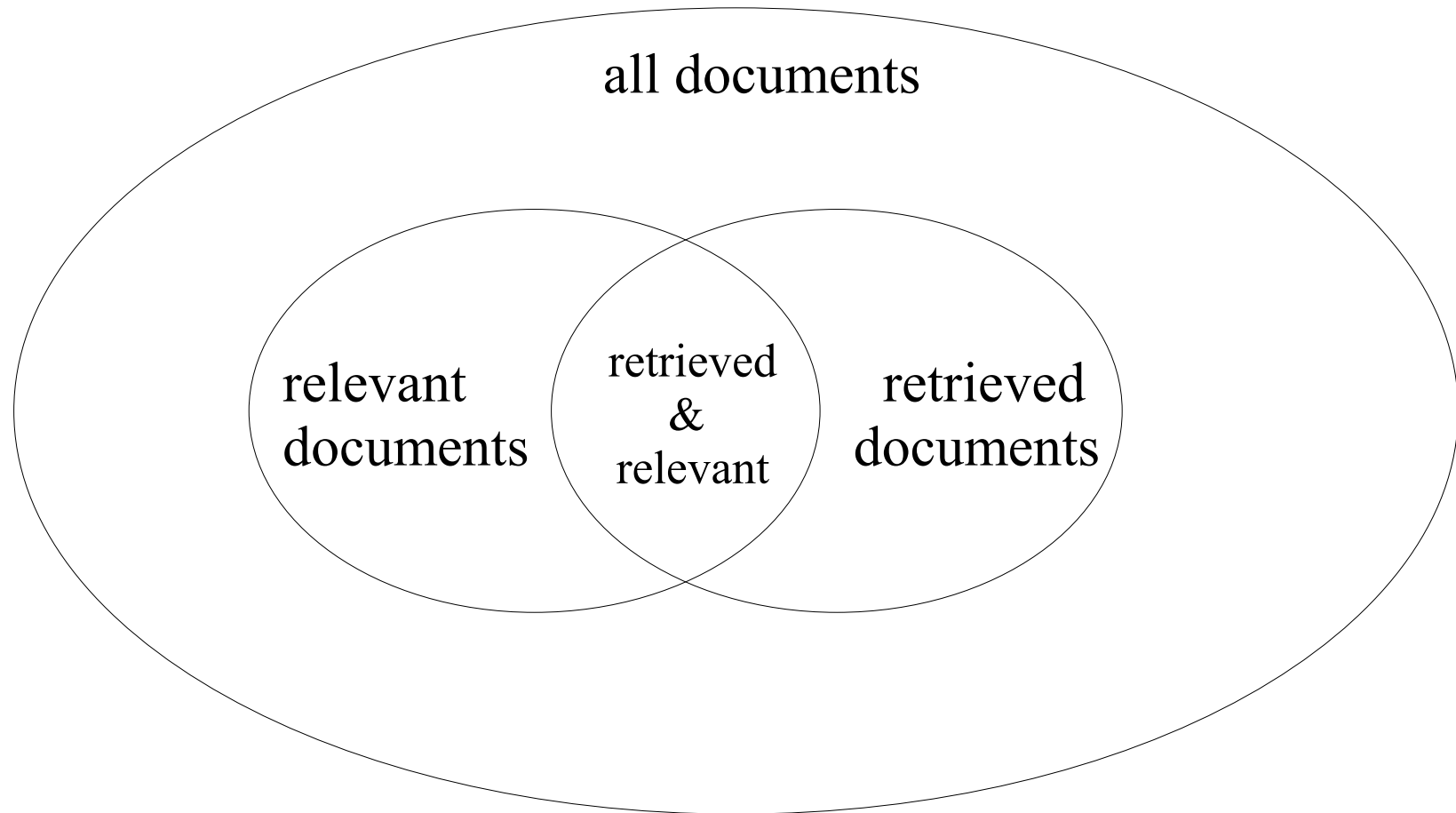
Relevance Ranking

- A query Q is represented as a document vector
- Compute similarity of document vector Q with all retrieved document vectors D
 - similarity is computed as the cosine of the **angle between the query vector and the document vector**

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

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

Evaluation of a Retrieval Result



Evaluation - Accuracy

- Confusion Matrix:

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

- $Accuracy =$ percentage of correctly retrieved documents

$$accuracy = \frac{a + d}{|D|}$$

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

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

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

F-Measure

- Weighted harmonic mean of recall and precision

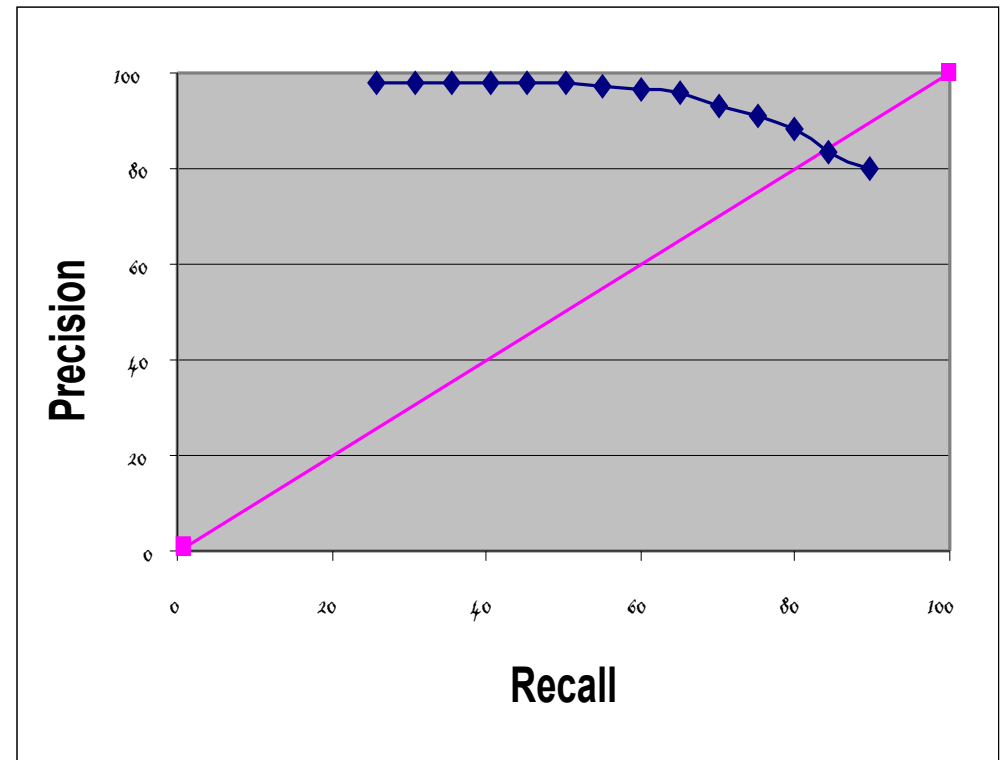
$$F_{\beta} = \frac{(\beta^2 + 1) pr}{\beta^2 p + r}$$

- equivalent form for $\alpha = \frac{\beta^2}{\beta^2 + 1}$: $F_{\alpha} = \frac{1}{\alpha \frac{1}{r} + (1 - \alpha) \frac{1}{p}}$
- 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 $q \in Q$ 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 (d_1, d_2, \dots, d_n)
 - compute a 0/1 relevance list (r_1, r_2, \dots, r_n)
 - $r_i = 1$ iff $d_i \in D_q$
 - $r_i = 0$ otherwise.

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

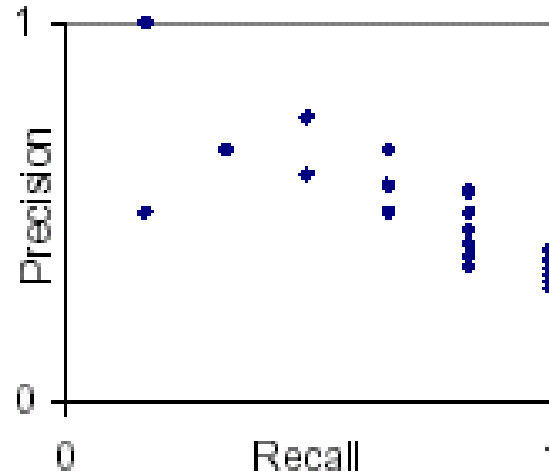
- compute $recall = \frac{1}{|D_q|} \sum_{1 \leq i \leq n} r_i$ and $precision = \frac{1}{n} \sum_{1 \leq i \leq n} r_i$

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 (d_1, d_2, \dots, d_k)
 - $recall(k) = \frac{1}{|D_q|} \sum_{1 \leq i \leq k} r_i$
- Precision at rank
 - Fraction of the top $k \geq 1$ responses that are actually relevant.
 - $precision(k) = \frac{1}{k} \sum_{1 \leq i \leq k} r_i$

Recall and Precision at Rank

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



Precision at rank plotted against recall at rank for the given relevance vector.

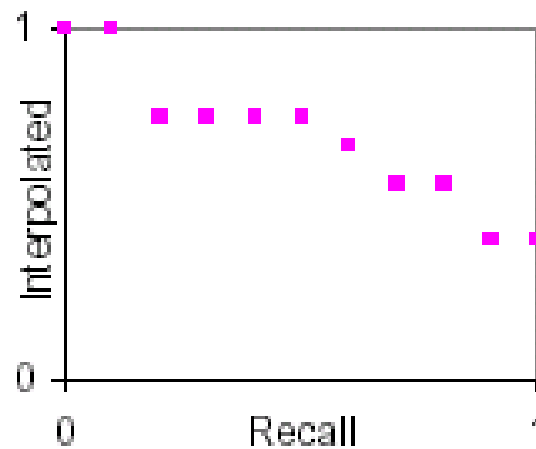
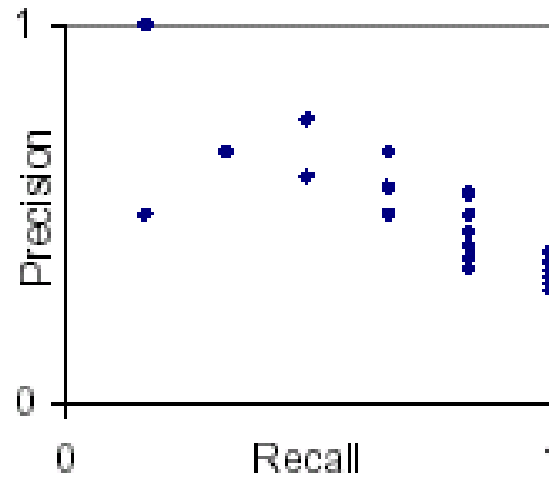
Missing r_k are zeroes.

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 precision 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	
3	1
4	1
5	
6	1
7	
8	
9	1
10	
11	
12	
13	
14	
15	1
16	
17	
18	
19	
20	

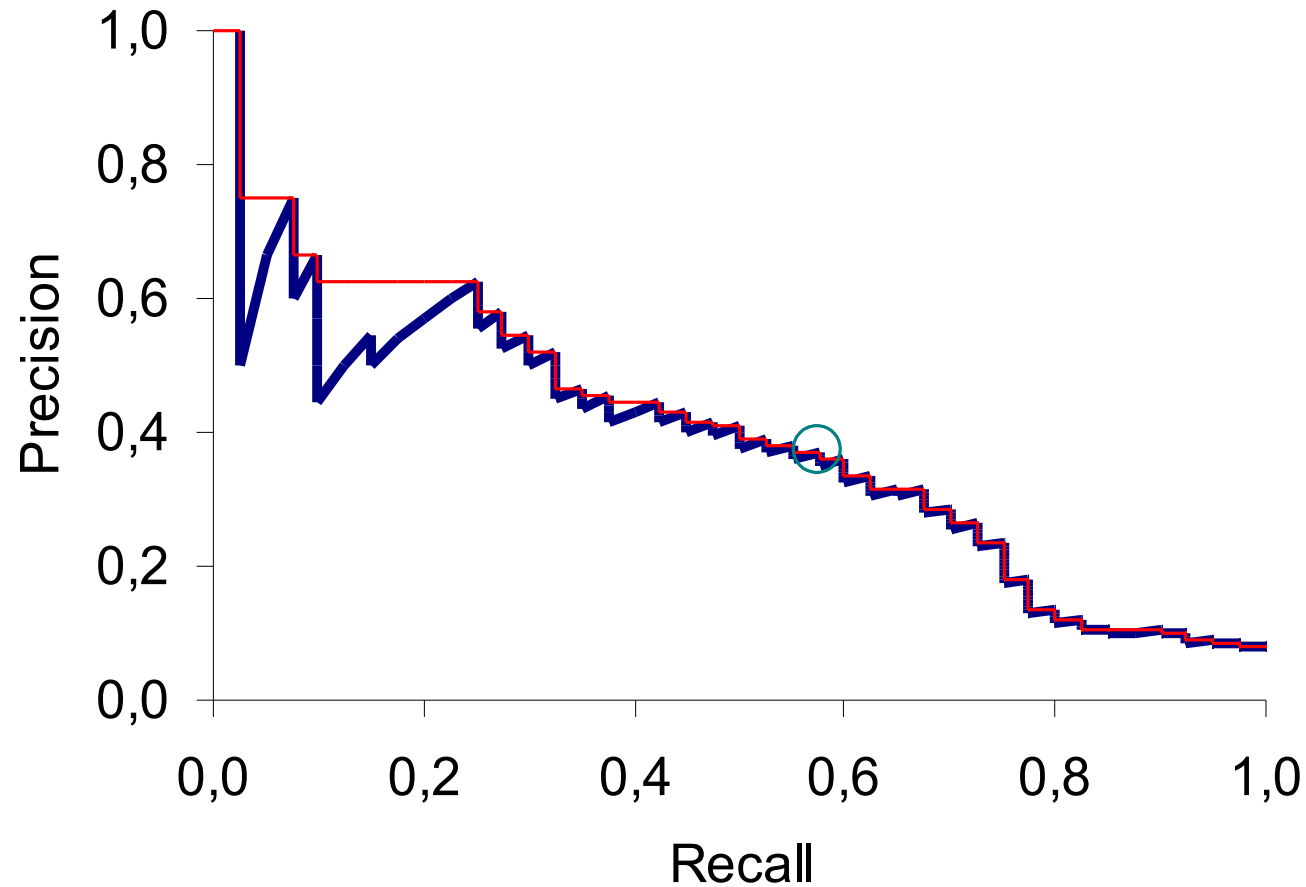


Precision and interpolated precision plotted against recall for the given relevance vector.

Missing r_k are zeroes.

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

$$avg. precision = \frac{1}{|D_q|} \sum_{1 \leq k \leq |D|} r_k \cdot precision(k)$$

- $avg. precision = 1$ iff
 - engine retrieves all relevant documents and
 - ranks them ahead of any irrelevant document

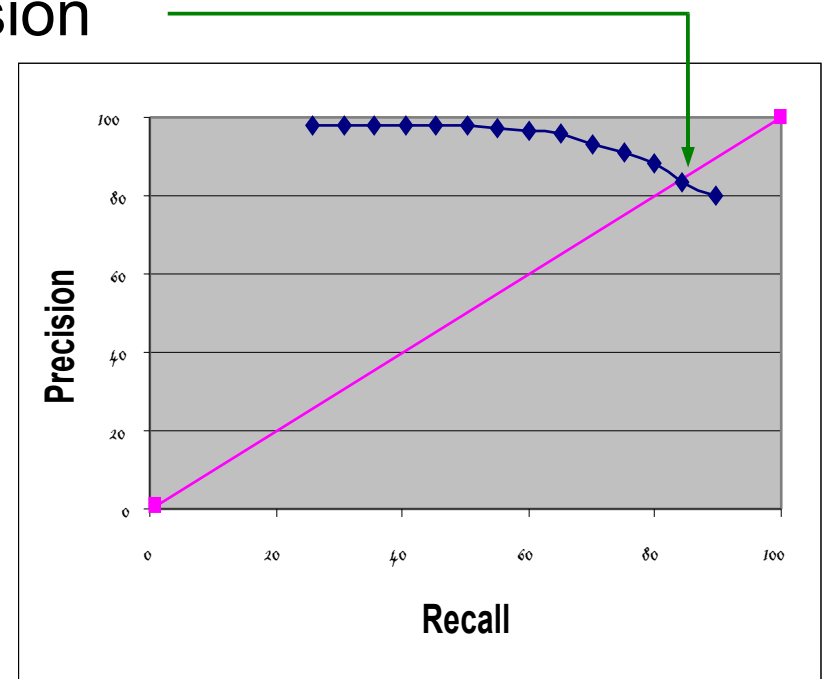
- 11point Average Precision

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

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



$$r = p = \frac{r_2 \times p_1 - r_1 \times p_2}{r_2 - r_1 + p_1 - p_2} = \mathbf{91,76}$$

Normalized Discounted Cumulative Gain (NDCG)

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

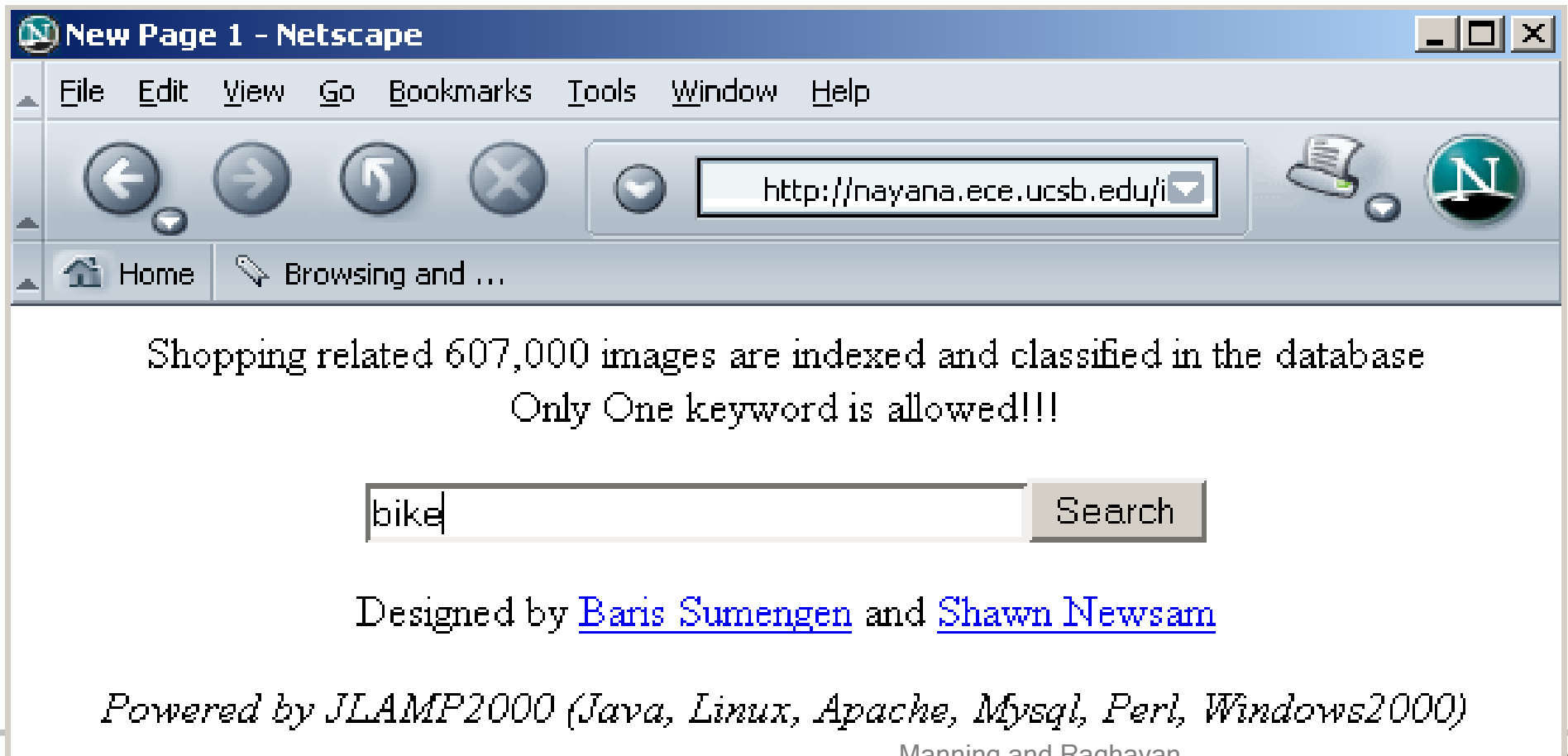
Relevance Feedback

1. Present an initial query result
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. Enrich original query with query terms from relevant documents
 - e.g., select by log odds ratio, add relevant document vectors, subtract irrelevant document vectors
4. Goto 1.


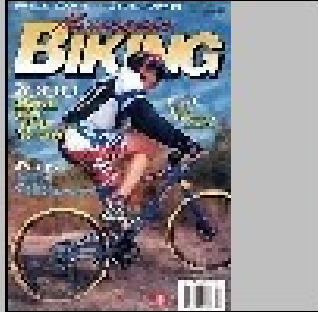










PROBLEM: increased effort for the user, only feasible for long-term monitoring of interactions (e.g., WebWatcher)

Relevance Feedback: Example













- Image search engine
<http://nayana.ece.ucsb.edu/imsearch/imsearch.html>



Results for Initial Query

					
(144473, 16458) 0.0 0.0 0.0	(144457, 252140) 0.0 0.0 0.0	(144456, 262857) 0.0 0.0 0.0	(144456, 262863) 0.0 0.0 0.0	(144457, 252134) 0.0 0.0 0.0	(144483, 265154) 0.0 0.0 0.0
					
(144483, 264644) 0.0 0.0 0.0	(144483, 265153) 0.0 0.0 0.0	(144518, 257752) 0.0 0.0 0.0	(144538, 525937) 0.0 0.0 0.0	(144456, 249611) 0.0 0.0 0.0	(144456, 250064) 0.0 0.0 0.0

Relevance Feedback

					
(144473, 16458) 0.0 0.0 0.0	(144457, 252140) 0.0 0.0 0.0	(144456, 262857) 0.0 0.0 0.0	(144456, 262863) 0.0 0.0 0.0	(144457, 252134) 0.0 0.0 0.0	(144483, 265154) 0.0 0.0 0.0
					
(144483, 264644) 0.0 0.0 0.0	(144483, 265153) 0.0 0.0 0.0	(144518, 257752) 0.0 0.0 0.0	(144538, 525937) 0.0 0.0 0.0	(144456, 249611) 0.0 0.0 0.0	(144456, 250064) 0.0 0.0 0.0

Results after Relevance Feedback

Browse

Search

Prev

Next

Random



(144538, 523493)
0.54182
0.231944
0.309876



(144538, 523835)
0.56319296
0.267304
0.295889



(144538, 523529)
0.584279
0.280881
0.303398



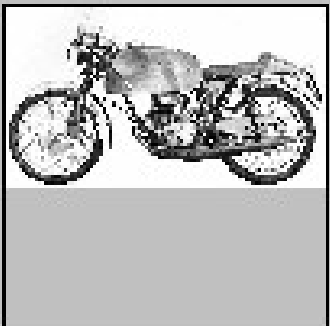
(144456, 253569)
0.64501
0.351395
0.293615



(144456, 253568)
0.650275
0.411745
0.23853



(144538, 523799)
0.66709197
0.358033
0.309059



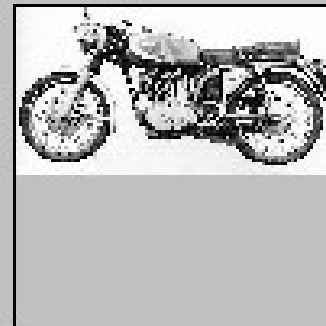
(144473, 16249)
0.6721
0.393922
0.278178



(144456, 249634)
0.675018
0.4639
0.211118



(144456, 253693)
0.676901
0.47645
0.200451



(144473, 16228)
0.700339
0.309002
0.391337



(144483, 265264)
0.70170796
0.36176
0.339948



(144478, 512410)
0.70297
0.469111
0.233859

Rocchio Algorithm

- increase weight of terms that appear in relevant documents R_j
- decrease weight of terms that appear in irrelevant documents I_j

$$Q_{i+1} = \alpha Q_i + \beta \sum_j R_j - \gamma \sum_j I_j$$

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

Example: Vivísimo

<http://www.vivísimo.com>



[company](#) | [products](#) | [solutions](#) | [customers](#) | [demos](#) | [press](#)

[Advanced Search](#) [Help](#)

Search [Clusty.com](#) with our **NEW FireFox Toolbar**

Clustered Results

- ▶ [jaguar](#) (223)
 - ▶ [Jaguar Cars](#) (36)
 - ▶ [Club](#) (32)
 - ▶ [Parts](#) (22)
 - ▼ [Panthera onca](#) (18)
 - ▶ [Animal](#) (8)
 - ▶ [Behavior, Discusses Physical](#) (4)
 - ▶ [Jaguar Facts](#) (3)
 - ▶ [Other Topics](#) (5)
 - ▶ [Models](#) (16)
 - ▶ [Sale, Buy](#) (14)
 - ▶ [Jaguar Owners](#) (14)
 - ▶ [Jag](#) (11)
 - ▶ [Neu Und Gebrauchte](#) (2)
 - ▶ [Restoration](#) (10)
 - ▼ [More](#)

Find in clusters:

Top 223 results of at least 10,240,000 retrieved for the query jaguar ([Details](#))

- ▶ [Jaguar bei mobile.de](#) [new window] [preview] Sponsored Link
Wunschauto finden, einfach & bequem Alle Modelle - neu und gebraucht!
www.mobile.de
- ▶ [Jaguar Top-Marken](#) [new window] [preview] Sponsored Link
Neu und gebraucht! Kostenloser Käuferschutz bis €400
www.ebaymotors.de
- 1. [Jaguar - Official Site](#) [new window] [frame] [preview] [clusters]
Covers available models, owner benefits such as roadside care and extended warranties, **Jaguar** history and press releases.
www.jaguarcars.com - Looksmart 1, MSN 2, Wisenut 3
- 2. [Jag-lovers](#) [new window] [frame] [preview] [clusters]
Enthusiasts can find a wealth of information about this make of automobile, including what to look at when buying a new or used Jag, forums, news, and related links.
www.jag-lovers.org - Open Directory 2, Lycos 8, Ask Jeeves 8, MSN 9, Looksmart 12
- 3. [Jaguar Cars](#) [new window] [frame] [preview] [clusters]
...rldwide web site of **Jaguar** Cars...
www.jaguar.com - Lycos 1, MSN 1, Ask Jeeves 1, MSN 31
- 4. [Apple - Mac OS X](#) [new window] [frame] [preview] [clusters]
Learn about the new OS X Server, designed for the Internet, digital media and workgroup management. Download a technical factsheet.
www.apple.com/macosex - Wisenut 2, MSN 4, Looksmart 23
- 5. [Jaguar - Classic Jaguar](#) [new window] [frame] [preview] [clusters]
Offers an index of parts and accessories, **Jaguar** news and a customer support FAQ. Check out the list of cars for sale.
www.classicjaguar.com - Looksmart 4, Wisenut 14, Ask Jeeves 15, MSN 42

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 **Yellow Pages** White Pages

Web Pages Images Audio Video News

 Exact Phrase

[Advanced Web Search](#)
[Preferences](#)
[Tools & Tips](#)

Now Searching: Google • Yahoo! • Ask Jeeves • About • Overture • Findwhat • [Learn More](#)

Web results for "jaguar" (1 - 20 of 94)

1 | 2 | 3 | 4 | 5 Next >

View Results by: Relevance Search Engine

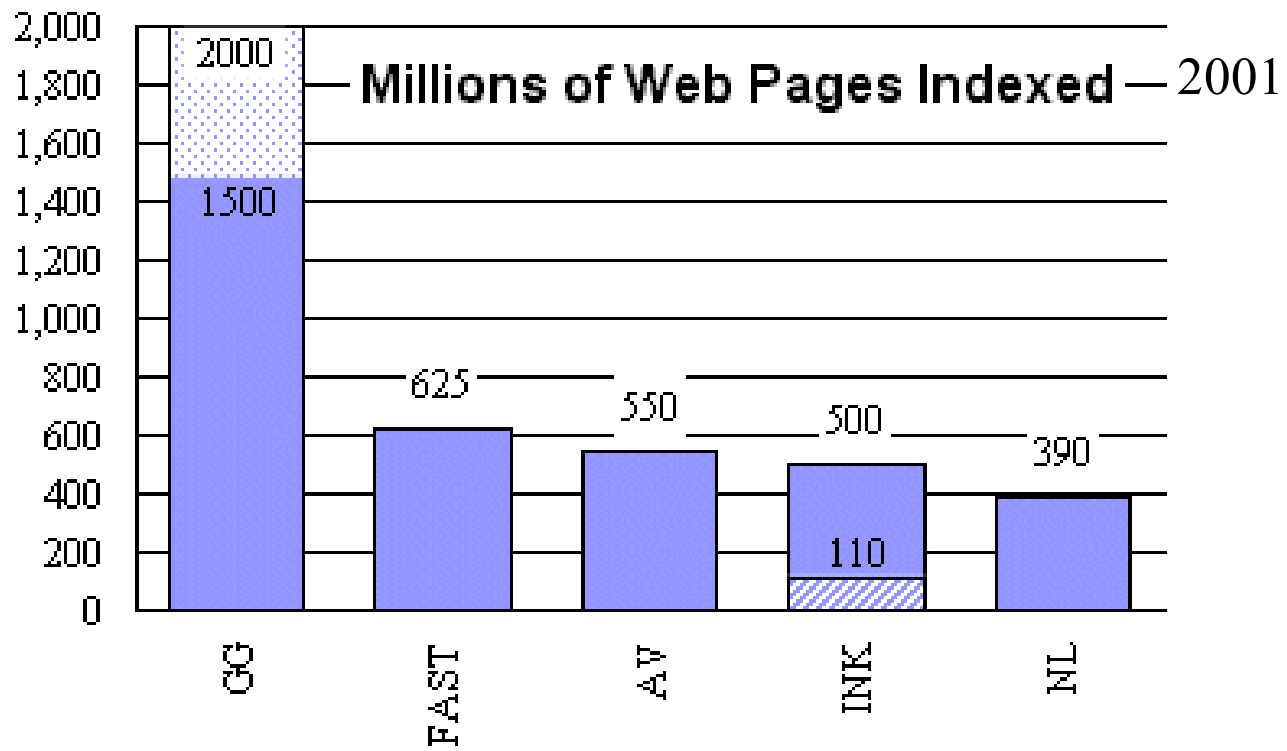
Are you looking for?

1. [Jaguar Cars](#)
Official worldwide web site of **Jaguar** Cars.
<http://www.jaguar.com/> [Found on Google, Yahoo!, Ask Jeeves]
2. [Jaguar Cars](#)
Click here to be redirected to www.jaguar.com.
<http://www.jaguarcars.com/> [Found on Google, Yahoo!, LookSmart Reviewed]
3. [Jag-lovers - THE source for all Jaguar information](#)
The Premier **Jaguar** Cars web resource for all enthusiasts.
<http://www.jag-lovers.org/> [Found on Google, Yahoo!, Ask Jeeves]
4. [Jaguar bei mobile.de](#)
Wunschauto finden, einfach & bequem Alle Modelle - neu und gebraucht!
Sponsored by: <http://www.mobile.de/> [Found on Web Search Picks]
5. [Jaguar bei eBay](#)
Riesen Auswahl zu Niedrigpreisen Mitbieten oder Sofortkaufen!
Sponsored by: <http://www.ebaymotors.de/> [Found on Web Search Picks]
6. [Apple - Mac OS X](#)
The Apple Mac OS X product page. Describes features in the current version of Mac OS X, a screenshot gallery, latest software downloads, and a directory of ...
<http://www.apple.com/macOSX/> [Found on Google, Yahoo!]
7. [Jaguar UK - R is for Racing](#)
... Le Mans winning C-TYPE - the first car ever to have disc brakes - Jaguar's racing technology has been bred into the bloodline of every Jaguar, ...
<http://www.jaguar-racing.com/> [Found on Google, Yahoo!]

[Jaguar Animal](#)
[Jaguar the Rainforest Animal](#)
[Jaguar Cars](#)
[Ferrari](#)
[Cheetah](#)
[Cat Jaguar](#)
[BMW](#)
[Leopard](#)

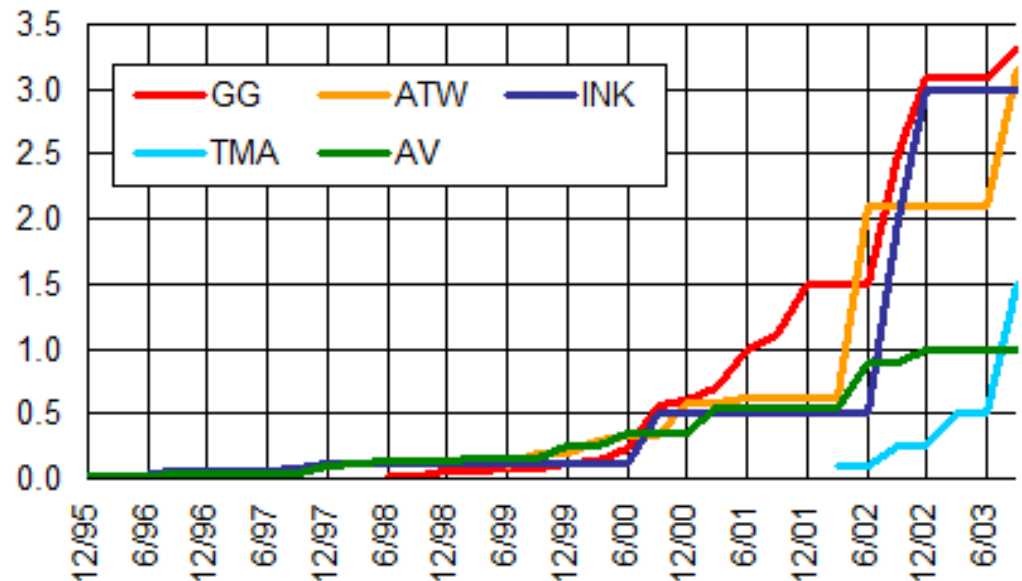
Web Search Engines

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



Search Engine Sizes

Search Engine Sizes
(millions of web pages)



Source:
searchenginewatch.com

Searches per Day

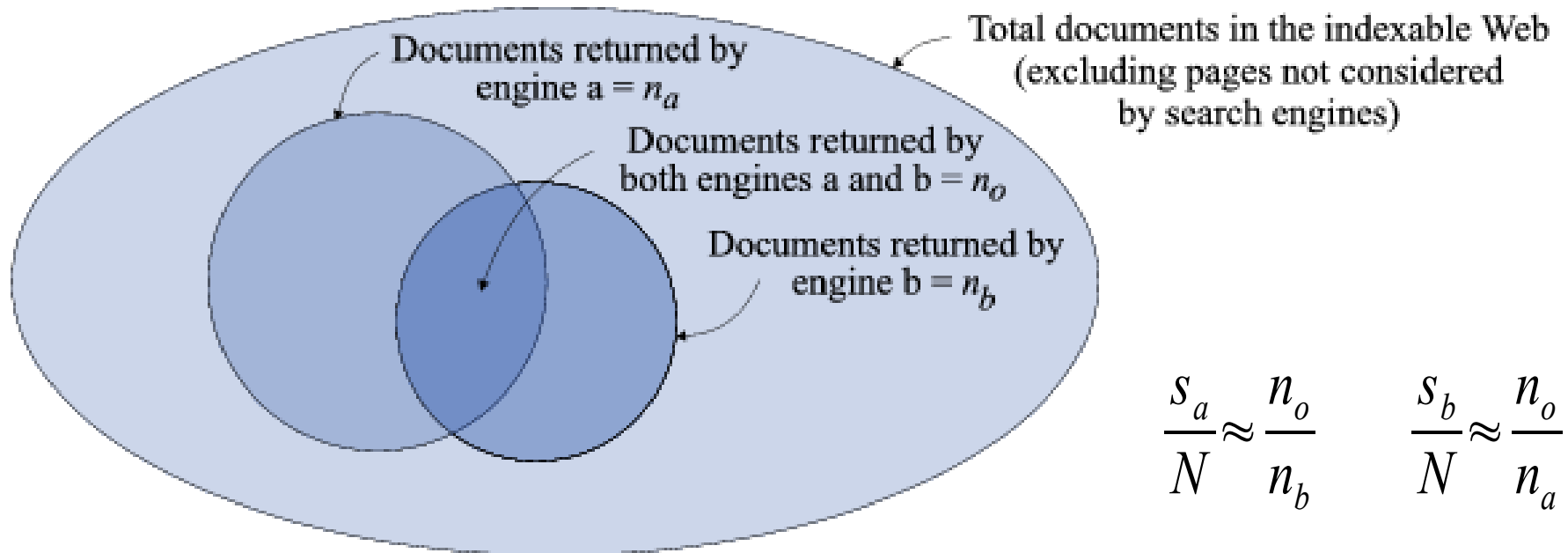
Service	Searches Per Day	As Of/Notes
Google	250 million	February 2003 (as reported to me by Google, for queries at both Google sites and its partners)
Overture	167 million	February 2003 (from Piper Jaffray's The Silk Road for Feb. 25, 2003 and covers searches at Overture's site and those coming from 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

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)



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

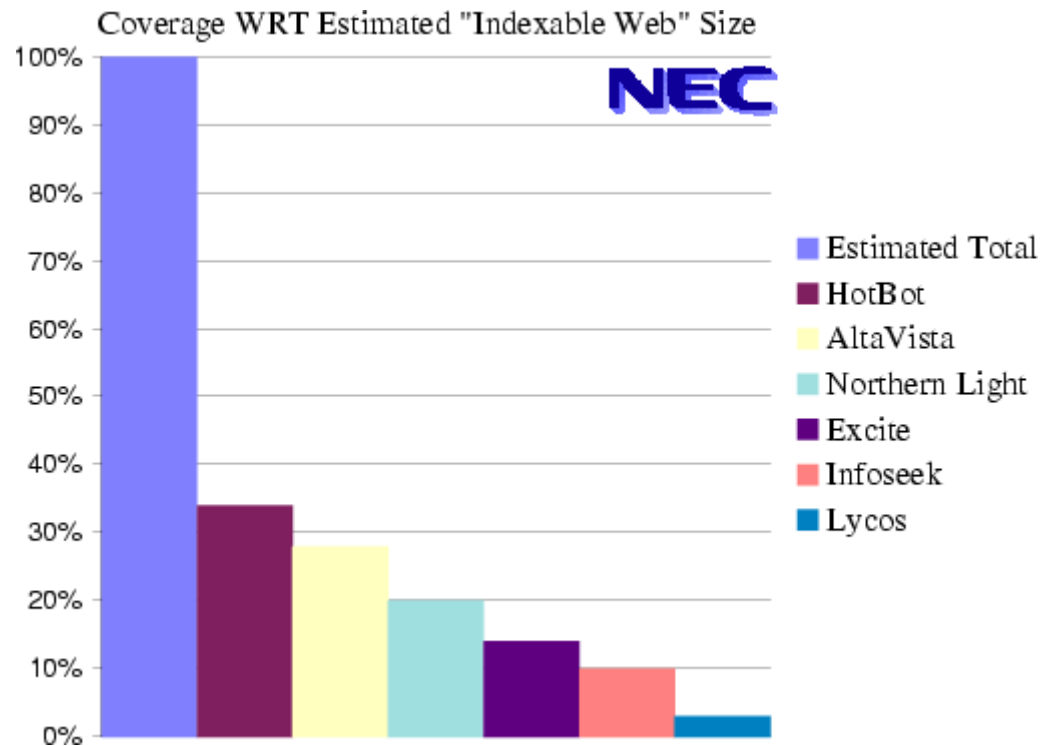
s_a = total number of pages indexed by search engine a
 s_b = total number of pages indexed by search engine b

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

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

Search Engine Coverage

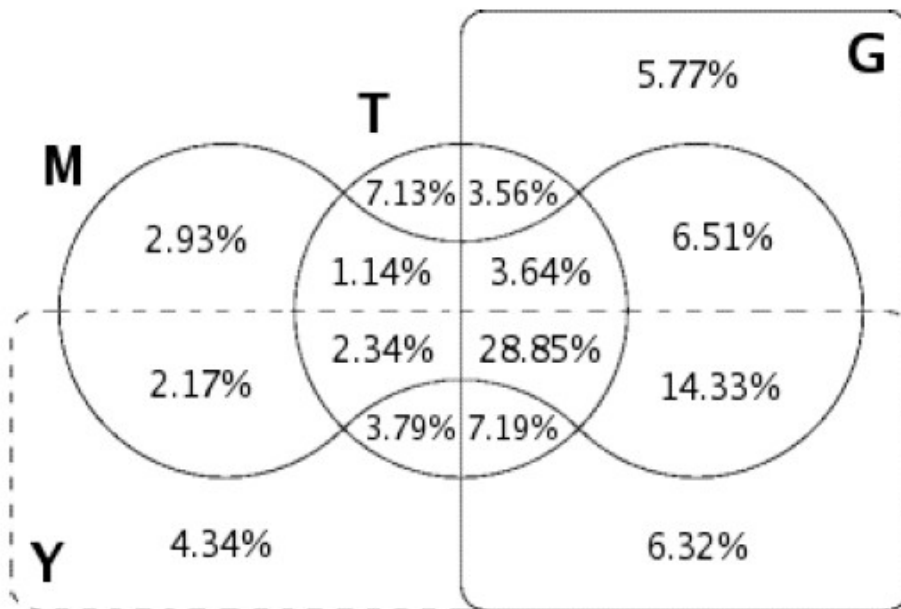
- Based on the estimated size of the Web 1998
- The best engine indexes only 32% of the "indexable Web"



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

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/>