A Survey On Web Information Retrieval Technologies

Lan Huang
Computer Science Department
State University of New York at Stony Brook

Presenter: Chen Yi-Ting

Reference：

Lan Huang, A Survey On Web Information Retrieval Technologies.(2000)

Outline

- Introduction
- Web Information Retrieval
- General-purpose Search Engines
- Hierarchical Directories and Automatic Categorization
- Measuring the Web
- Conclusion

Introduction

- This report explores the engineering details and algorithmic issues behind Web information retrieval systems.
  - To compare Web retrieval and classical information retrieval and show where the challenges are
  - To review the representative search engine and their architectural features
  - To present the engineering issues in building a robust search engine and the existing algorithms developed to provide high-quality and high-relevance results
  - To describe the Codir system which is designed to solve the online update problem
  - To discuss the algorithms , architecture and performance of the automatic classification system
  - The problem of web statistics collection is discussed together with interesting results from an analysis of AltaVista query log

Web Information Retrieval

- The uniqueness of Web information retrieval：
  - Bulk
  - Dynamic Internet
  - Heterogeneity
  - Variety of Language
  - Duplication
  - High Linkage
  - Ill-formed queries
  - Wild Variance in Users
  - Specific Behavior

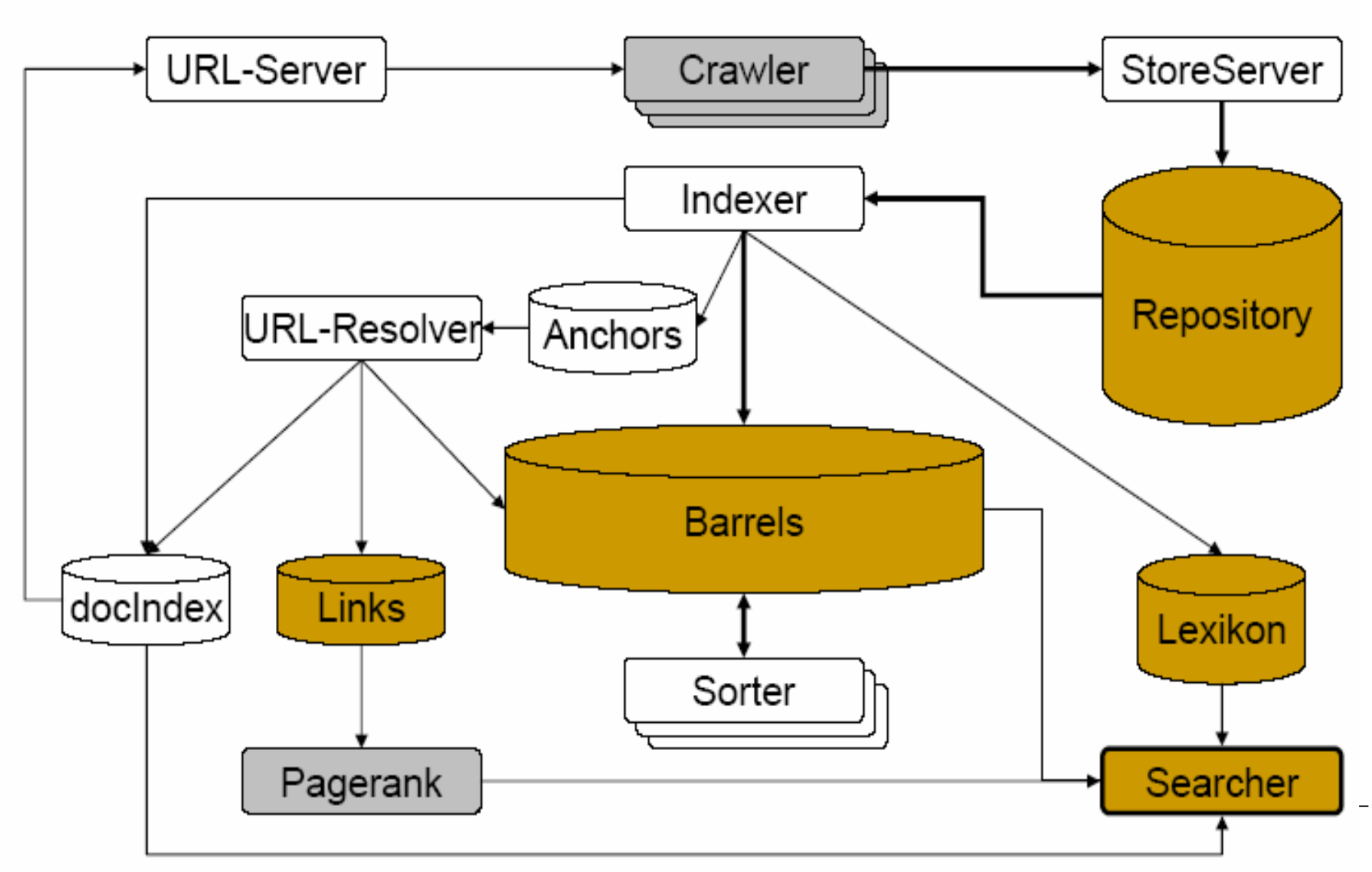
- The big challenges to Web information retrieval is to meet the users information needs given the heterogeneity of the Web and the ill-formed queries

General-purpose Search Engines

- The Goal is to return both high-relevance and high-quality (in other word, valuable) pages.
- Current Status of Search Engines—
  - Five most popular search engines：Google、Alta Vista、Northern Light、InfoSeek、FastSearch
- Architecture of A Search Engine
  - Architecture
  - Data Structure
- Engineering Issues (**for building a robust search engine**)
  - Crawling the Web
  - Caching Query Results
  - Incremental Updates to Inverted Index
- Algorithmic Issues (**for providing a high-quality IR service**)
  - Ranking
  - Duplicate Elimination

Architecture of A Search Engine

Architecture



Architecture of A Search Engine

- The web crawler
  - URLserver
  - Storesserver
- The indexing function
  - The indexer
    - Read & Uncompress docs from Respository, and parses them
    - Each document is converted into a set of word occurrence called hits
    - To distribute these hits into a set of “barrels”, creating a partially sorted forward index
    - Anchors file and URLresolver
  - The sorter
    - To take the barrels, which are sorted by docID and resorts them by wordID to generate the inverted index (need little temporary space)
    - DumpLexicon takes this list together with the lexicon produced by the indexer and generates a new lexicon to be used by the search
- The searcher
  - It is run by a web server
  - Use the lexicon with the inverted index and the PageRanks to answer queries

Architecture of A Search Engine

Data Structure (1/4)

- Repository
  - Contains the full HTML text of every web page
  - Each page is compressed using zlib (RFC1950)
  - The documents are prefixed by docID, length, and URL

- Document Index
  - The information stored in each entry includes the current document status, a pointer into the repository, a document checksum, and various statistic
  - A pointer (docinfo-contains its URL and title / urllist)
  - A list of URL checksums：to convert URLs into docIDs
  - URLresolver：turn URLs into docIDs.
- Lexicon
  - Keep it in memory on a machine
  - The current lexicon contains 14 million words (Google)

Architecture of A Search Engine

Data Structure (2/4)

- Hit Lists
  - A hit list corresponds to a list of occurrence of a particular word in a particular document including position, font, and capitalization information
  - Google's writers chose a hand optimized compact encoding

Hit: 2 bytes

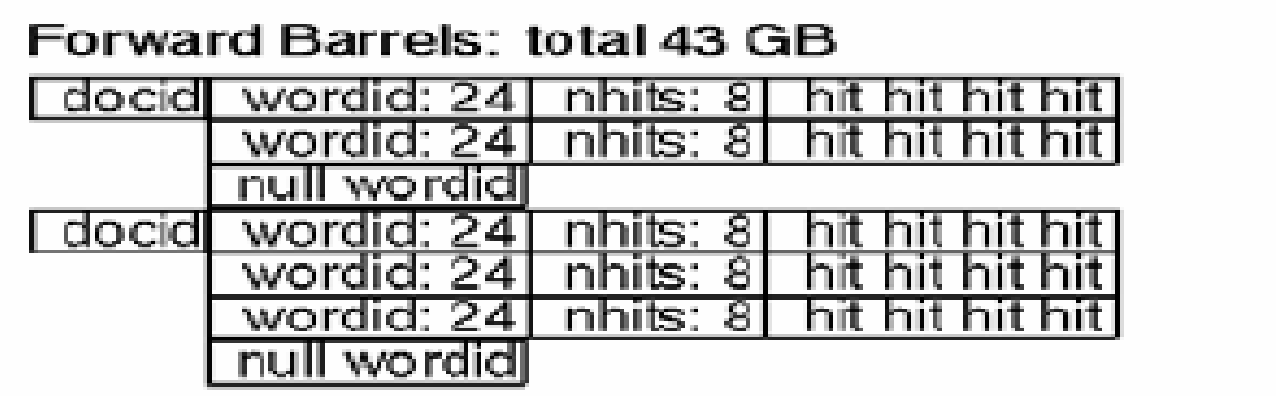
plain:	cap:1	imp:3	position: 12		
fancy:	cap:1	imp = 7	type: 4	position: 8	
anchor:	cap:1	imp = 7	type: 4	hash:4	pos: 4

  - There are two types of hits：plain hits and fancy hits (imp=111)
    - Fancy hits include hits occurring in a URL, title, anchor text, or meta tag
    - Plain hits include every thing else
    - Anchor hits

Architecture of A Search Engine

Data Structure (3/4)

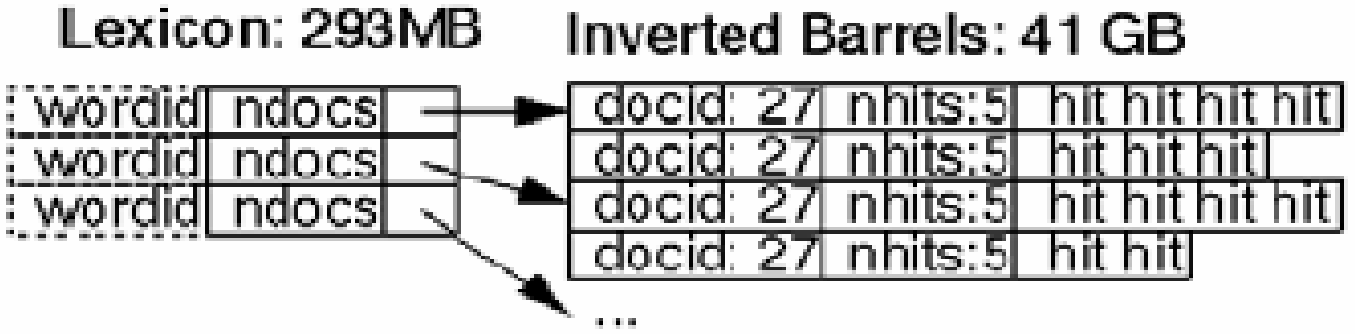
- Forward Index
  - The forward index is actually already partially sorted
  - The docID is recorded into the barrel, followed by a list of wordID's with hit lists which correspond to those words
  - Each barrel holds a range of wordID
  - Each wordID is stored as a relative difference from the minimum wordID



Architecture of A Search Engine

Data Structure (4/4)

- Inverted Index
  - For every valid wordID, the lexicon contains a pointer into the barrel that wordID falls into
  - It points to a doclist of docID's together with their corresponding hit list
  - Importance issue is in what order the docID's should appear in the doclist
    - Sorted by docID (quick merging the doclists)
    - Sorted by a ranking of the occurrence of the word in each doc
    - Google chose a compromise (keep two sets of inverted barrels)



Engineering Issues

Crawling the Web (1/2)

- Google crawler
  - Externally, the crawler must avoid overloading Web sites or network links as it goes about its business
  - Internally, the crawler must efficiently deal with huge volumes of data
  - It must decides：in what order to scan the URLs in the queue, in what frequency to revisit pages to keep it up to date
  - Google has a fast distributed crawling system, and a single URLserver serves lists of URLs to a number of crawlers
    - Each crawler keeps roughly 300 connection open at once
    - As peak speeds , the system can crawl over 100 web pages per second using four crawlers
  - A mojor performance stress is DNS lookup (each crawler maintains its own DNS cache
  - Asynchronous I/O to manage events

Engineering Issues

Crawling the Web (2/2)

- Cho etc. 1999
  - To spread the work load among all crawlers in their crawling system
    - The crawler splits all URLS which are going to be crawled into 500 queues based on a hash of their server name
    - The crawlers read one URL from each queue at a time, moving to a new queue moving to a new queue for each URL
    - Only one connection is allowed from the crawler to a particular server at a time

Engineering Issues

Caching Query Results(1/2)

- Caching documents to reduce access delay is extensively used on the web
  - Most web browsers cache documents in the client's main memory or in some local disk
  - To improve cache hit rates, cache proxies are used
  - A query result caching policy for search engine site query cache is proposed[Markatos99]
    - There exists temporal locality (20%~30%) in the queries submitted
    - Two-stage LRU (LRU-2S) cache replacement policy
  - From the performance evaluation：
    - Medium-sized caches can easily exploit the locality fount and result in hit rates ranging from 25%~30% (a few hundred Mbytes large)
    - Effective Cache Replacement Policies should take into account both recency and frequency of access in their replacement decision

Engineering Issues

Caching Query Results(2/2)

Initial state of the Queue:

primary		secondary	
4	3	2	1

After accessing 1:

1	4	3	2

After accessing 4:

4	1	3	2

After accessing 5:

4	1	5	3

After accessing 5:

5	4	1	3

After accessing 6:

5	4	6	1

Engineering Issues

Incremental Updates to Inverted Index (1/5)

- To discuss previous efforts in information retrieval to speed up incremental update, and to discuss Codir system which supports online update
- Callan94 (INQUERY system)
  - Using the Mnenme persistent object store to manage its inverted file index
    - This system with only a small impact on query processing
  - Over 90% of the inverted lists are less than 1000bytes (in their compressed form), and account for less than 10% of the total inverted file size
  - Nearly half of all lists are less than 16 bytes
  - This means that many inverted lists will never grow after their initial creation

Engineering Issues

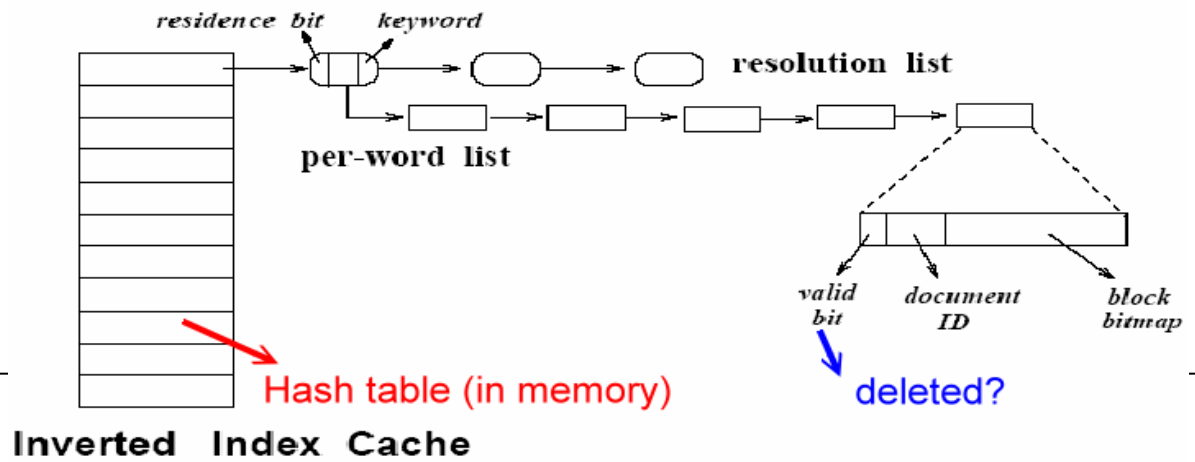
Incremental Updates to Inverted Index (2/5)

- Callan94 (INQUERY system)
  - The main allocation scheme：
    - Lists are allocated using a range of fixed size objects (range from 16 to 8192 bytes by power of 2)
    - When a new list is created, an object of the smallest size large enough to contain the list is allocated
    - A list can then grow to fill the object. When it exceeds the object, a new object of the next larger object size is allocated (then copy and free)
    - When a list exceeds the largest object size, a linked list of 8192 byte objects is started
- Garcia-Molina94
  - Propose a new data structure that manage small inverted list in buckets and dynamically select large inverted lists to be managed separately.
- Cutting and Pederson 90
  - Optimizations for dynamic update with a B-tree

Engineering Issues

Incremental Updates to Inverted Index (3/5)

- Falustsos and Jagadisk
  - HYBRID scheme
  - Provides a number of parameters to control the size of the chunks and the length of chains
- To provide non-stopping search engine service：
  - keep a second copy (with update operation)
  - Can't update & search simultaneously
- Codir (Author's system L.Huang 98)—a way that update and searching are serviced simultaneously on the same set of inverted lists



Engineering Issues

Incremental Updates to Inverted Index (4/5)

- Codir (Author's system L.Huang 98)

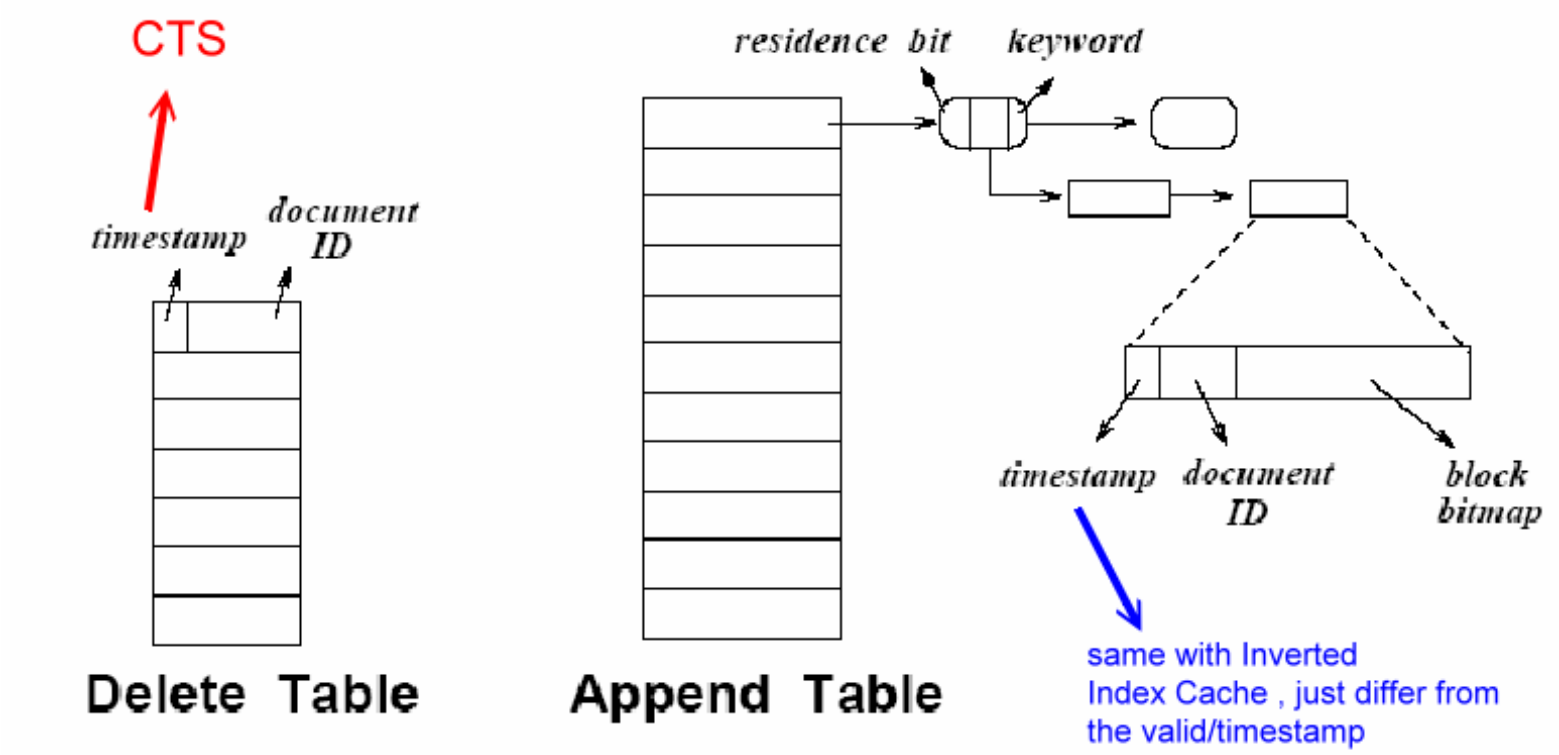


Figure 3: Data Structure Used in Codir

Engineering Issues

Incremental Updates to Inverted Index (5/5)

- Codir (Author's system L.Huang 98)
  - At any point in time , only a subset of the inverted index is memory resident
  - Query request
    - Search the inverted list cache
      - If miss, the corresponding inverted list is loaded
    - Combine the list with Append Table
    - Before return , scan the Delete Table & mark the deleted docID (maximum CTS as CWTS[current working timestamp])
    - Locking mechanism for inverted list (multi-thread)
  - Append、Delete Table are reflected into the permanent storage periodically

Algorithmic Issues

Ranking

- PageRanking Algorithm
  - Notation
    - A has pages T1… Tn which point to it (citations)
    - d range from 0~1, a damping factor (google set 0.85)
    - C(A)：number of links going out of page A
  - The probability that the random surfer visits a page is its PageLink (the d factor)

$$PR(A) = (1-d) + d\left(PR(T_1)/C(T_1) + ... + PR(T_n)/C(T_n)\right)$$

  - High PageRank
    - Many pages pointing to it
    - Or there are some pages that point to it and have a high PageRank

Algorithmic Issues

Ranking- **HITS Algorithm**(1/4)

- HITS Algorithm
- Given a query , HITS will find good sources of content (Authorities) and good source of links (hubs)
  - Authorities
    - Large in-degree
  - Hub
    - Pull together authorities on a given topic (Like Yahoo)
  - A mutually reinforcing relationship：A good hub is a page that points to many good authorities; a good authority is a page that is pointed to by many good hubs

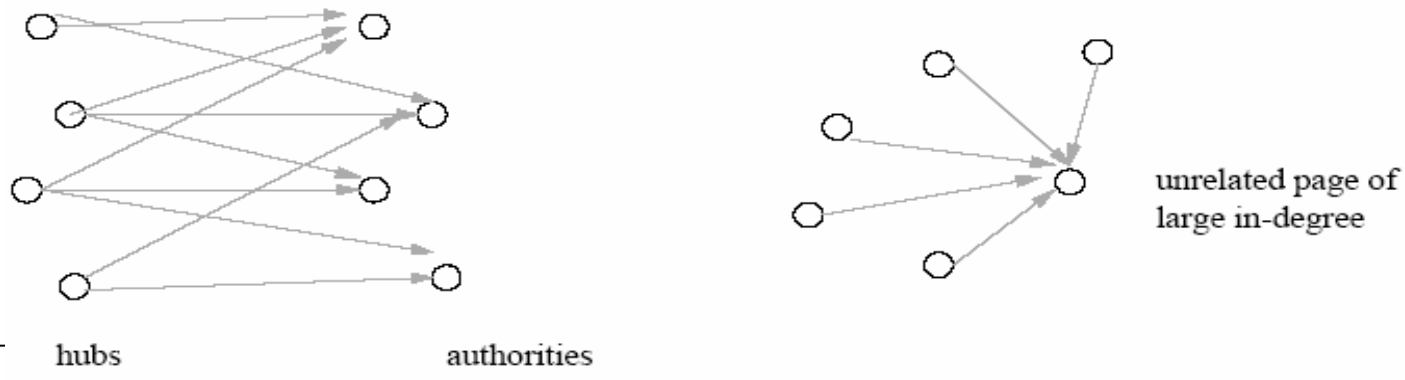


Figure 4: A densely linked set of hubs and authorities

Algorithmic Issues

Ranking- **HITS Algorithm**(2/4)

- Considering the Web structure
  - page = node
  - link = directed edge
- To decide the set of pages that will work on
  - Subset of all Web pages
  - Non-trivial algorithms– high cost
    - By ensuring it is rich in relevant pages
  - Set of pages (S_σ) with special properties
    - S_σ is relatively small
    - S_σ is rich in relevant pages
    - S_σ contains many of the strongest authorities

Algorithmic Issues

Ranking- **HITS Algorithm**(3/4)

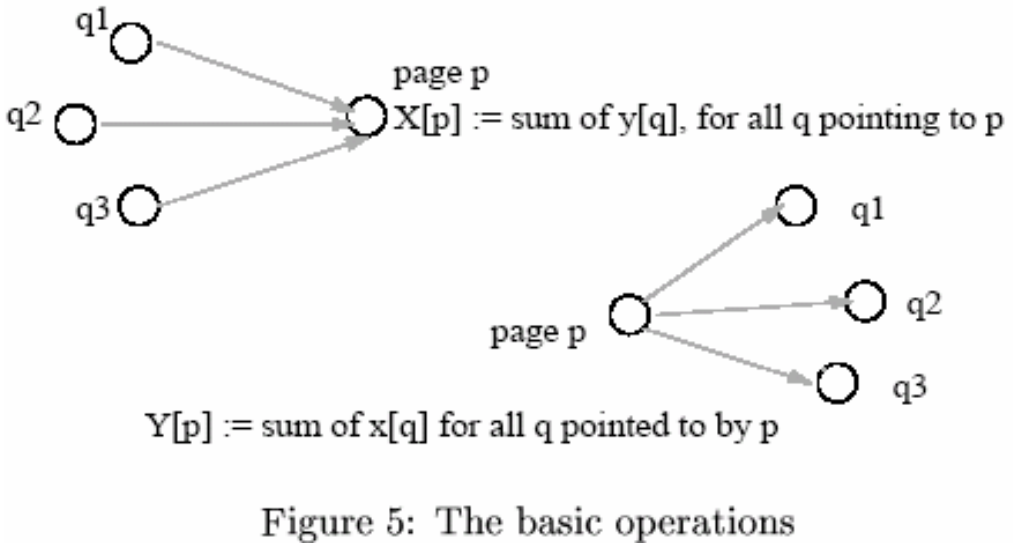


Figure 5: The basic operations

For i = 1, 2, ... k
Apply the $\mathcal{I}$ operation to (x_{i-1},y_{i-1}), obtaining new x-weights x'_i.
Apply the $\mathcal{O}$ operation to (x_i',y_{i-1}), obtaining new y-weights y'_i.
Normalize x'_i, obtaining x_i.
Normalize y'_i, obtaining y_i.
End
Return (x_k,y_k).

- An Iterative Algorithm
  - $\mathcal{I}$ operation updates the x-weights
  - $\mathcal{O}$ operation updates the y-value as follows
  - If p points to many pages with large x-value, then it should receive a large y-value; and if p is pointed to by many pages with y-values, then it should receive a large x-value

$$x^{<P>} \leftarrow \sum_{q:(q,p)\in E} y^{<q>}$$

$$y^{<P>} \leftarrow \sum_{q:(q,p)\in E} x^{<q>}$$

Algorithmic Issues

Ranking- **HITS Algorithm**(4/4)

- Bharat and Henzinger point out that HITS didn't work well in all cases due to the following three reasons：
  - Mutually Reinforcing Relationship Between Hosts
  - Automatically Generated Links
  - Non-Relevant Node
- Krshna Bharat and Monika Henzinger present two basic approaches to tackle topic drift
  - Eliminating non-relevant node from the graph
  - Regulating the influence of a node based on its relevance
    - K edges➔1/k authority weight
    - L edges➔1/l hub weight

$$A[n] := \sum_{(n',n)\in N} H[n'] \times auth\_wt(n',n)$$

$$H[n] := \sum_{(n',n)\in N} A[n'] \times hub\_wt(n',n)$$

Algorithmic Issues

Ranking- **Others**

- Anchor Text, Headings etc.
- Anchor text
  - Provide better quality reults
  - The large amounts of data must be processed
- Cutler (1997)
  - Assign different weight to heading as well as anchor text (help WebIR)
  - They group HTML tags into six classes: Plain text, Title, H1-H2, H3-H6, Strong and Anchor
  - Conclusion is that anchor texts and STRONG (STRONG、B、OL、UL) class should carry more weight

Algorithmic Issues

Duplicate Elimination(1/6)

- Approximately 30% of pages are (near) duplicates
- Challenges
  - Defining the notation of a replicated collection precisely
    - Slight differences between copies
  - Efficient algorithm to identify such collection and exploiting this knowledge of replication
    - Hundreds of millions of pages
  - One of the major difficulties in detecting replicated collection is that many replicas may not be strictly identical to each other：
    - Update Frequency
    - Mirror Partial Coverage
    - Different Formats
    - Partial Crawls

Algorithmic Issues

Duplicate Elimination(2/6)

- One can determine the similarity of two pages in a variety of way：
  - To use the information retrieval notion of textual similarity
  - To use data mining techniques to cluster pages into groups that share meaningful terms
  - To compute textual overlap by counting the number of common chunks of text that page share
- The formal definition for similar collection：
  - Equisized collection C1 and C2 are similar if there is a one-to-one mapping M that maps all C1 pages to all C2 pages
    - **Similar pages**
    - **Similar links**

Algorithmic Issues

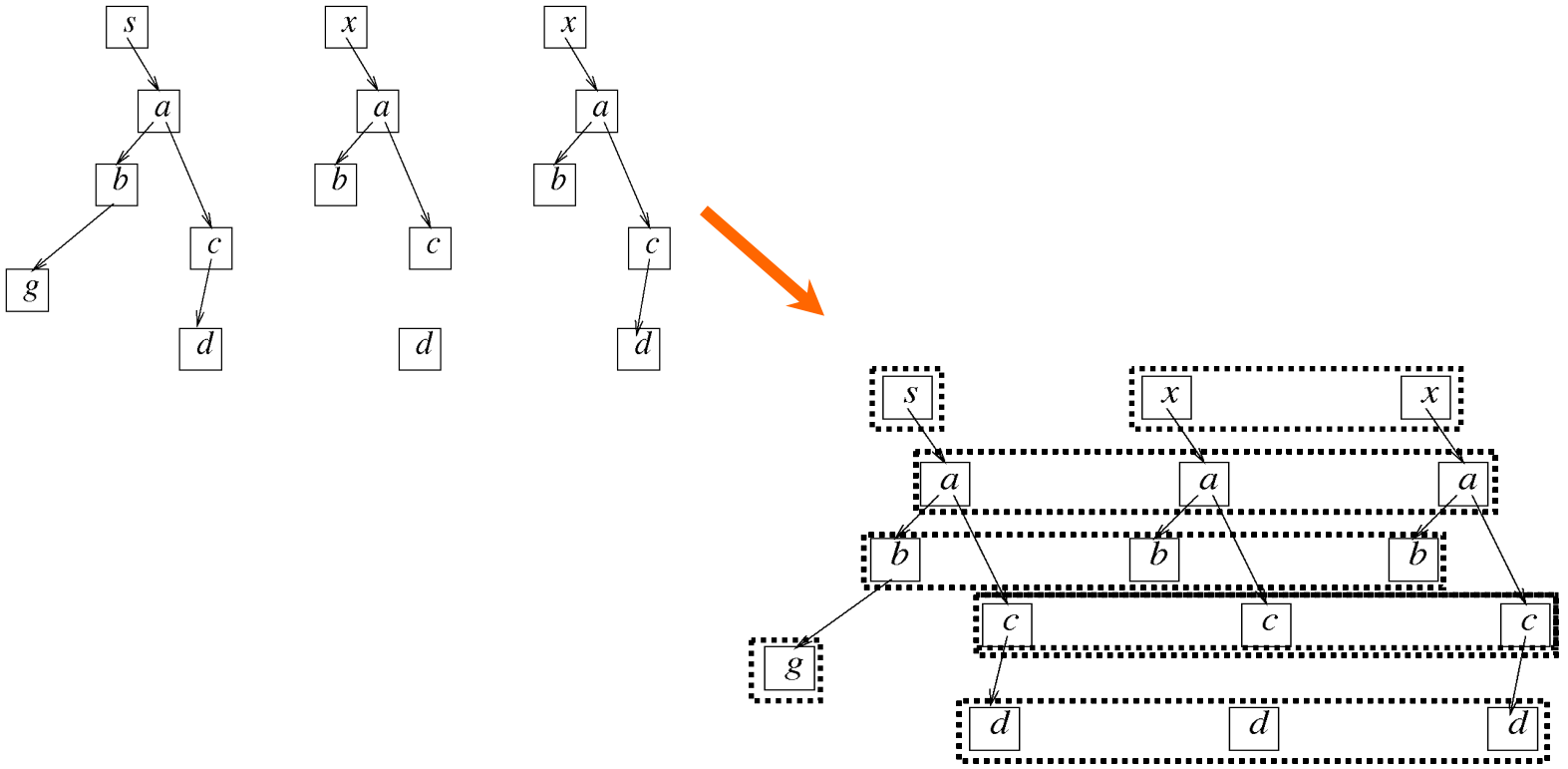
Duplicate Elimination(3/6)

- Growth Strategy
  - Identify similar cluster within a given web graph
  - It grows cluster from smaller sized ones
- Definitions:
  - Cluster
    - A set of equi-sized collections to be a cluster
  - Identical Cluster
    - All its collection are identical
  - Similar Cluster
    - All of its collections are pairwise similar

Algorithmic Issues

Duplicate Elimination(4/6)

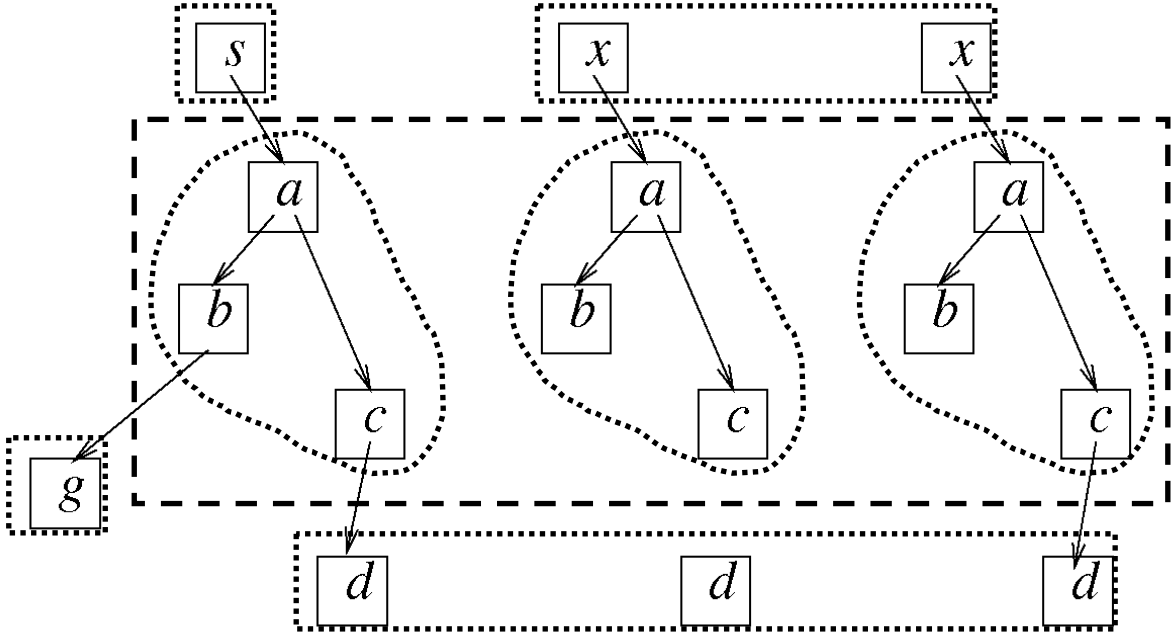
- Growth strategy
  - First , to compute trivial cluster on the given graph



Algorithmic Issues

Duplicate Elimination(5/6)

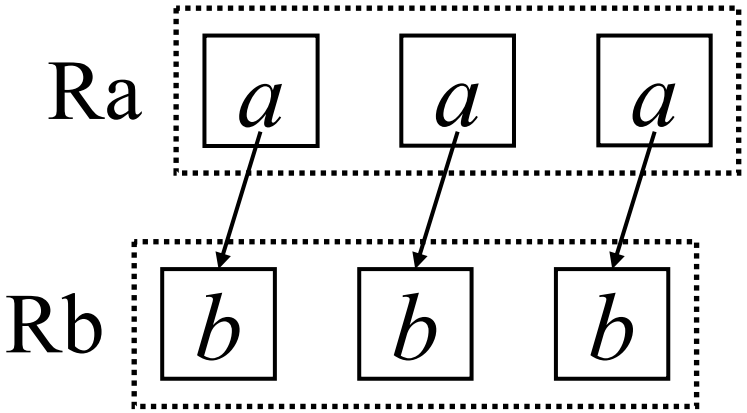
- Growth strategy
  - Next, to merge trivial clusters that can lead to similar cluster with larger collections



Algorithmic Issues

Duplicate Elimination(6/6)

- Essential property (Merge condition)



Ls: # of pages linked *from*

Ld: # of pages linked *to*

$$|Ra| = Ls = Ld = |Rb|$$

Hierarchical Directories and Automatic Categorization

- Hierarchical directories are portals to the Web for Internet users
- Current Status of Hierarchical Directories
  - Open Director Project (open directory source)

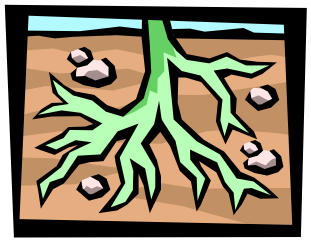
name	Librarians' Index	Infomine	Britannica Web's Best	Yahoo!	Galaxy
Size,type	About 5,000. Compiled by public librarians in information supply business. Highest quality sites only. Great annotations.	About 16,000. Compiled by academic librarians.	About 150,000. Hand-picked, annotated, and ranked by Britannica editors.	About 1 million. Scarce descriptions and annotations. Biggest and most famous directory around. Many sub-Yahoo's by region, country, topic.	About 300,000. Generally good annotations.
Phrase searching	No.	Yes. Use " "	Yes. More than word searched as phrase.	Yes. Use " "	No.
Boolean logic	AND implied between words. Also accepts OR and NOT	AND implied between words. Also accepts OR.	Accepts AND, OR, NOT	No.	OR implied between words. Also accepts AND, OR, NOT
Sub-Searching	No.	No.	In results, specify SORT by subject in result.	Yes. In results, select search within category or all of Yahoo.	No.

Table 2: Most Popular Directories(Nov,1999)

Automatic Categorization -Taper(1/10)

- Taper
  - A taxonomy-and-path-enhanced-retrieval system
  - In a data base that associated not only keywords but also topics with documents
  - Goal
    - Construct a classifier
    - Apply to new documents
  - Context-sensitive features
    - A function (signature) of both the document and the topic path (context)

Automatic Categorization -Taper(2/10)



Class Taxonomy

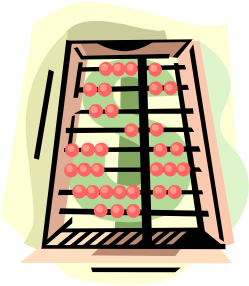
Training Documents

class | doc id

… | …

Class-doc Relation

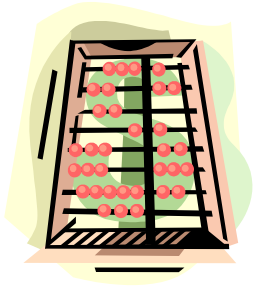
Training System



Statistics

Feature Terms

Automatic Categorization -Taper(3/10)



Statistics

Feature Terms

Incoming Documents

Classification System

Target Class

Automatic Categorization -Taper(4/10)

- Statistics Collection
  - The goal of this module is to collect term statistics from a document and dispenses with it as fast as possible
  - A term is a 32-bit ID, which could represent a word, a phrase, words from a linked docs, etc.
- Feature Selection
  - Find the best feature to discriminate the document from another
  - Find the optimal subset of terms out of large lexicon terms appears impractical
  - The Taper, it first orders the terms by decreasing ability to separate the class

$$score(t) = \frac{\text{Interclass distance}}{\text{Intraclass distance}} = \frac{\sum c_1, c_2 (\mu(c_1,t) - \mu(c_2,t))^2}{\sum_c \frac{1}{|c|} \sum_{d \in c} (f(t,d,c) - \mu(c,t))^2}$$

f(t,d,c)=the number of times term t occurs in doc d in the training set of class c

c, c_1, c_2 ：children of internal node c_0 $\quad \mu(c,t) = \frac{1}{|c|} \sum_{d \in c} f(d,c,t)$

Automatic Categorization -Taper(5/10)

- Evaluation
  - Suppose c_0 has children $\langle c_1, \ldots, c_n \rangle$ given a class model (Bernoulli model , each face of the coin corresponding to some term t) , the classifier at estimate the parameters for each child.
  - When a new doc is input , a document in class is generated (using the class models and Byes's law)
  - Native Bayes' law
    - Estimates the conditional probability of the class given the document

$$P(c \mid d, \theta) = \frac{P(d \mid c, \theta) P(c \mid \theta)}{P(d \mid \theta)} \propto P(d \mid c, \theta) P(c \mid \theta)$$

    - θ - parameters of the model
    - *P(d)* – normalization factor $(\Sigma_c P(c|d)=1)$
    - Assumption: the terms in a document are conditionally independent given the class

Automatic Categorization -Taper(6/10)

- Evaluation
  - For classification we choose the class c that maximizes the following a priori class probability based on the Bernoulli model

$$\Pr[d \in c \mid c_0, F] = \frac{\text{(prob of d in c) * (prob of t in class c)}^{\text{times t occurred in d}}}{\text{Sum of numerator for all classes c} = \{\ c_1, \ldots c_l\ \}}$$

- $$= \frac{\pi(c)\prod_{t \in d \cap F}\theta(c,t)^{n(d,t)}}{\sum_{c'}\pi(c')\prod_{t \in d \cap F}\theta(c',t)^{n(d,t)}}$$

  - F：top F features
  - ：the prior prob. of class c
  - $\pi(c)$ ：prob. that “face” t turns up , estimated using f(f,d,c)
  - $\theta(d,t)$：num of times term t occurred in doc d

Automatic Categorization -Taper(7/10)

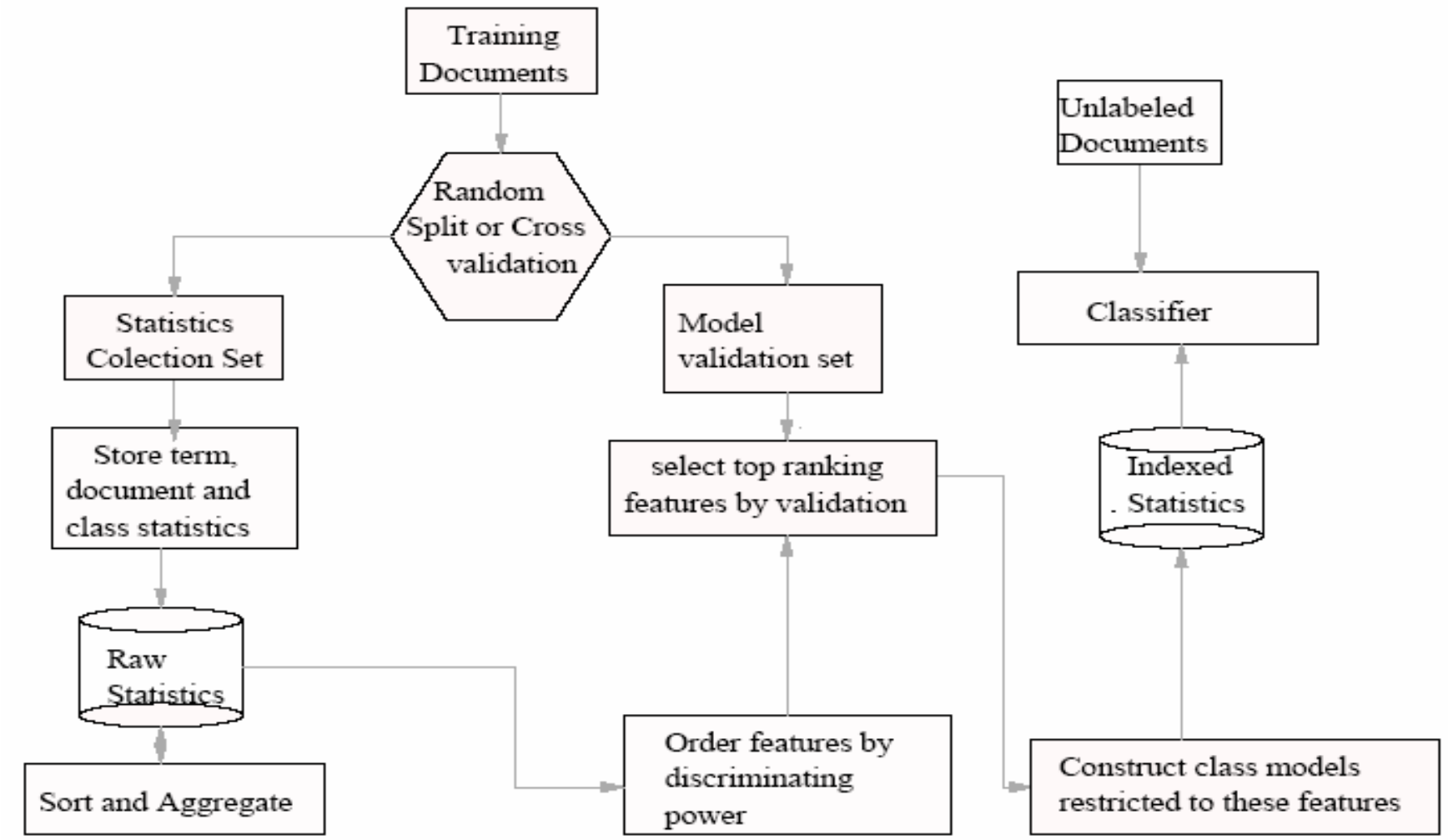


Figure 7: A sketch of the TAPER hierarchical feature selection and classification engine

Automatic Categorization -Taper(8/10)

- Enhanced Categorization Using Hyperlinks
  - Links in hypertext contain high-quality clues
  - Simply adding terms from neighbor texts will make error rate even higher
  - Notation

Δ= corpus = {documents}= $\{\delta_i$, i = 1,2,...n$\}$

$i \rightarrow j$ = direct link

G(Δ) = graph of linked documents

A(G)= adjacency matrix = $\{a_{i,j}\}$, $a_{i,j}$ = 1 if i $\rightarrow$ j link exists

τ_i = {terms (text) of di} = $\{\tau_{ij}$, j= 1, 2, ... |di|$\}$

T= $\{\tau_i \in D\}$ = set of text-sets for the corpus

C = $\{c_i$, set of possible class assignments for Δ $\}$

Ni = {Ii, Oi } = in-neighbors and out-neighbors of δ_i

Automatic Categorization -Taper(9/10)

- Radius-one specialization
  - Bootstrap mechanism
    - 1. Classifying unclassified documents from neighborhood of using term-only classifier
    - 2. Then, use this information to classify
    - Iterative 1&2 until constraint

$$\Pr[C \mid G,T] = \frac{\Pr[C,G,T]}{\Pr[G,T]} = \frac{\Pr[G,T \mid C]\Pr[C]}{\Pr[G,T]}$$

➔ choosing C to maximize Pr(G,T|C)* Pr(C)

$$\Pr(G,T|C)*\Pr(C) = \Pr(N_i|C_i)*\Pr(C_i)$$

$$\Pr(N_i \mid C_i) = \prod_{\delta_j \in Ii} \Pr(C_j \mid C_i, j \to i) \prod_{\delta_k \in Oi} \Pr(C_k \mid C_i, j \to i)$$

Automatic Categorization -Taper(10/10)

- Radius-two specialization
  - Co-citation is well-studied in linked corpora such as academic papers
  - Bridges are common documents hinting that two or more pages belong to the same class, without committing what that class could be
  - An “IO-bridge” connects to many pages of similar topics

Automatic Categorization -OpenGrid and ODP

- Manual categorization faces the scalability problem.
- ODP (Open Directory Project)
  - Allows thousands of volunteers who are familiar with some specific topics to classify sub-directories.
  - A centralized system
  - Rank homepages as cool pages and not-so-cool
- M. Lifantsev proposed a solution (OpenGrid system)
  - Distributed system utilizing all potential web surfers' opinions and not restricted to number of registered volunteers as ODP.
  - Extension of HTML
    - Classifying field , named cat
    - A field indicating evaluation of the page
  - To utilize the thousands of surfers' opinion and comments on the pages to rank the documents in the
  - no system is running yet.

```
<A href="http://www.somenews.foo/" cat="/News/Computers" rank="80%">
Good computer news</A>
```

Measuring the Web(1/9)

- Typical Questions
  - How big is the web? how fast does the Web grow?
  - How do various search engines compare?
  - Bharat and Broder described a technique to compare the coverage of different search engines
- Approach
  - Measure search engine coverage and overlap through random queries
  - Allows a third party to measure relative sizes and overlaps of search engines
  - Take two search engines, E1 and E2, we can:
    - Compute their relative sizes
    - Compute the fraction of E1's database indexed by E2

Measuring the Web(2/9)

- Procedures for Implementation
  - **Sampling**: A procedure for picking pages uniformly at random from the index of a particular engine $A \cap B$
  - **Checking**: A procedure for determining whether a particular page is indexed by a particular engine

- **Overlap Estimate**
  - the fraction of E1's database indexed by E2 is estimated by:

    Fraction of URLs sample from E1 found in E2
- **Size Comparison**
  - for search engines E1 and E2, Size(E1)/Size(E2) is estimated by：

$$\frac{\text{Fraction of URLs sample from E2 found in E1}}{\text{Fraction of URLs sample from E1 found in E2}}$$

Measuring the Web(3/9)

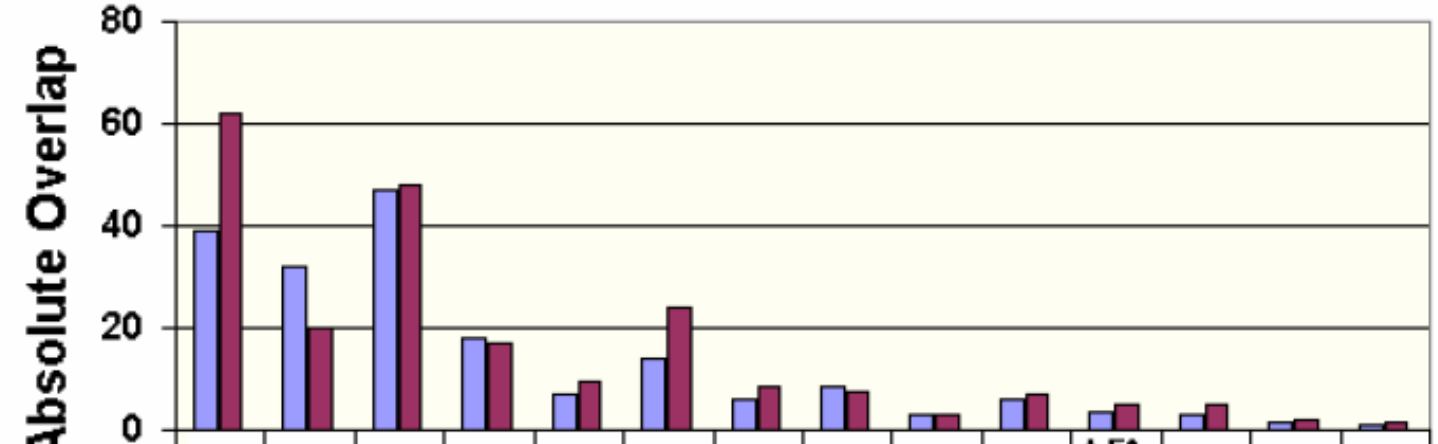
- Implementation
  - Building the Lexicon
  - Query based sampling
    - A random URL is generated by using a random query and randomly selecting a URL from the resulting set
    - Random selection of URL is only chosen from the first 100 results
    - Experiments are performed with both disjunctive and conjunctive queries
  - Query based checking
    - To test whether a search engine has indexed a given URL, we construct strongly query to check
    - Actual Matching – this can be done multiple ways: **Full URL**, **high similarity**, **weak URL**, **non-zero set**

Measuring the Web(4/9)

- Bias
  - Query Bias – favors large content rich documents
  - Ranking bias – introduced by search engines ranking pages. Only subsets are served up by the search while the remaining pages are not sampled.
  - Checking Bias – the method of matching and policy towards dynamic and low content pages influence the probability of the samples

Measuring the Web(5/9)

- In November 1997 , only 1.4% of all URLs indexed by the search engines



	A	E	H	I	A&E	A&H	A&I	E&H	E&I	H&I	A,E&H	A,H&I	E,H&I	ALL
Trials 1&2	39	32	47	18	7	14	6.2	8.6	3.1	5.9	3.6	3.1	1.6	0.9
Trials 3&4	62	20	48	17	9.3	24	8.7	7.5	2.9	6.8	5.2	4.9	1.8	1.4

Figure 8: Normalized estimates for all intersections (expressed as a percentage of total joint coverage) where A-AltaVista, I-Infoseek, E-Excite, H-HotBot

Measuring the Web(6/9)

- November 1997, AltaVista claims a coverage of 100 million pages and seems to have indexed roughly 50% of the web

➔conclude ：the static portion of the web is about 200 million pages

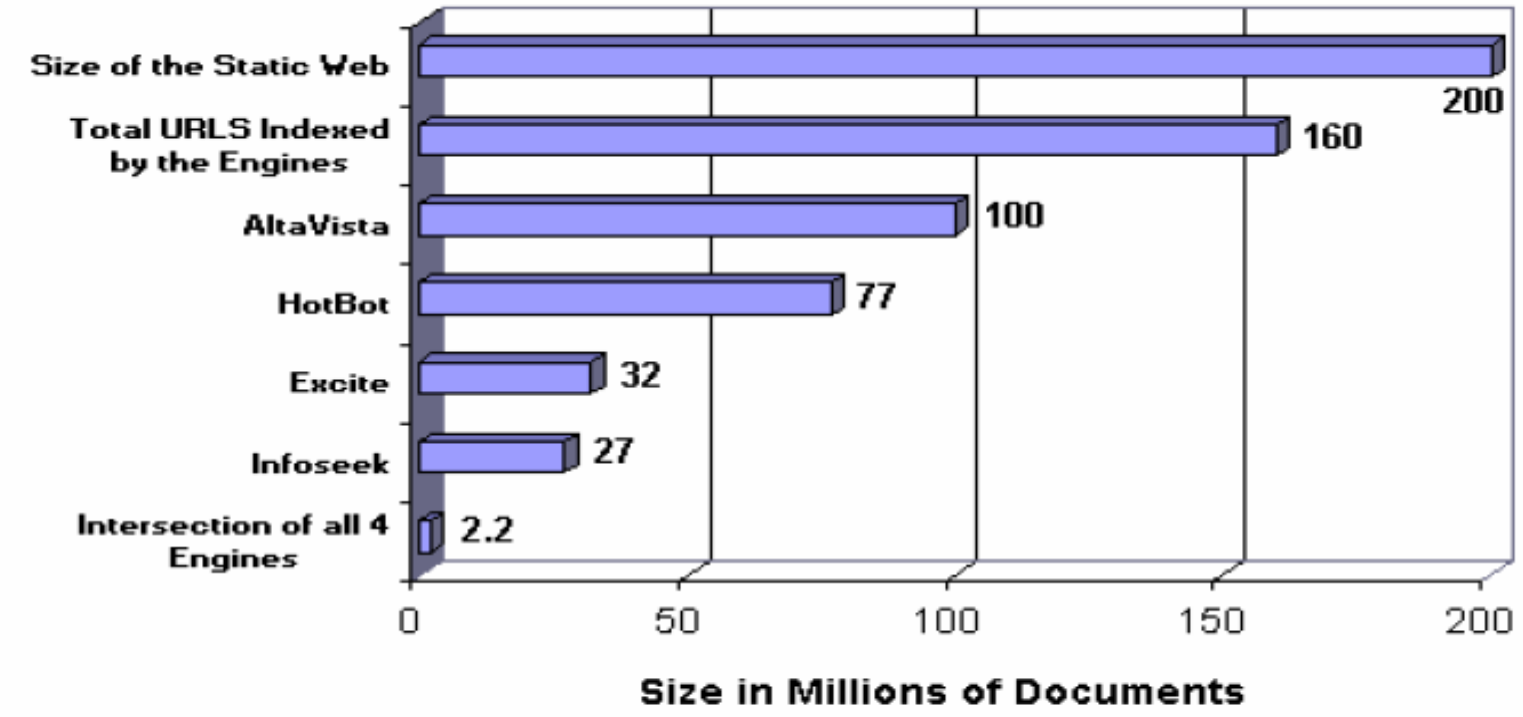


Figure 9: Absolute size estimates for November 1997.

Measuring the Web(7/9)

- Silverstein (1998)
  - Analysis of a very large AltaVista query log
    - Web users type in short queries , mostly look at the first 10 results only, and seldom modify the query.
    - Highly correlated items are constituents of phrases.
    - Fully 15% of all request were empty.
    - Of the non-empty requests, 32% consisted of a request for a new result screen, while 68% consisted of a request for the first screen of a new query.

Total number of bytes	300,210,000,000
Total number of requests	993,208,159
Total number of non-empty requests	843,445,731
Total number of non-empty queries	575,244,993
Total number of unique,non-empty queries	153,645,050
Total number of sessions	285,474,117
Total number of exact-same-as-before requests	41,922,802

Table 3: Statistics summarizing the query log contents used in the experiments. Empty requests had no query terms. A request consists of either a new query or a new requested result screen. Exact-same-as-before requests had the same query and requested result page as the previous request. The total number of non-empty, unique queries gives the cardinality of the set consisting of all queries.

Measuring the Web(8/9)

- Table4&5 summarize the statistics concerning the terms and operators in single query

0 terms in query	20.6%	max terms in query	393
1 terms in query	25.8%	avg terms in query	2.35
2 terms in query	26.0%	stddev of terms in query	1.74
3 terms in query	15.0%	>3 terms in query	12.6%

Table 4: Statistics concerning the number of terms per query Only distinct queries were used in the count; queries with many result screen requests were not up-weighted. The mean and standard deviation are calculated only over queries with at least one term.

0 operators in query	79.6%	max operators in query	958
1 operators in query	9.7%	avg operators in query	0.41
2 operators in query	6.0%	stddev of operators in query	1.11
3 operators in query	2.6%	>3 operators in query	2.1%

Table 5: Statistics concerning the number of operators – +, −, and, or, not, and near – per query. Only distinct queries were used in the count; queries with many result screen requests were not up-weighted.

Measuring the Web(9/9)

- Average number of queries per session is 2.02 and the average screens per query is 1.39

query occurs 1 time	63.7%	max query frequency	1,551,477
query occurs 2 times	16.2%	avg query frequency	3.97
query occurs 3 times	6.5%	stddev of query freq	221.31
query occurs > 3 times	13.6%		

Table 6: Statistics concerning how often distinct queries are asked. Only distinct queries were used in the count; queries with many result screen requests were not up-weighted. Percents are of the 154 million unique queries.

1 query per session	77.6%	max queries per session	172325
2 query per session	13.5%	avg queries per session	2.02
3 query per session	4.4%	stddev of queries/session	123.40
> 3 queries per session	4.5%		

Table 7: Statistics concerning the characteristics of query modification in sessions

1 screens per query	85.2%	max screens per query	78496
2 screens per query	7.5%	2nd most screens	5108
3 screens per query	3.0%	stddev of screens/query	1.39
> 3 screens per query	4.3%	avg screens per query	3.74

Table 8: Statistics concerning the characteristics of result screen requests in sessions