A survey on Web Information Retrieval Technologies

Lan Huang

Present: Yao-Min Huang

Date:03/02/2004、03/23/2004

Reference：

Ruslan Hristov：Authoritative Sources in a Hyperlinked Environment

Junghoo Cho： Finding Replicated Web Collections

Outline

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

Introduction

- First
  - Compare Web retrieval and classical information retrieval and show where the challenges are
- Second
  - Review the representative search engine and their architectural features
  - Describe the Codir system which is designed to solve the online update problem
- Third
  - Discuss the algorithms , architecture and performance of the automatic classification system
- Fourth
  - Analysis the query log

Web Information Retrieval

- The uniqueness of Web IR
  - Bulk
  - Dynamic Internet
  - Variety of Language
  - Duplication
  - High Linkage
  - Ill-formed queries
  - Wild Variance in Users
  - Specific Behavior
- Big challenge to Web IR
  - Heterogeneity of the Web
  - ill-formed queries

General-purpose Search Engines

- The Goal
- Current Status of Search Engines
- 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
    - PageRanking
    - HITS Algorithm
    - Others (Anchor Text , Headings etc.)
  - Duplicate Elimination

The Goal

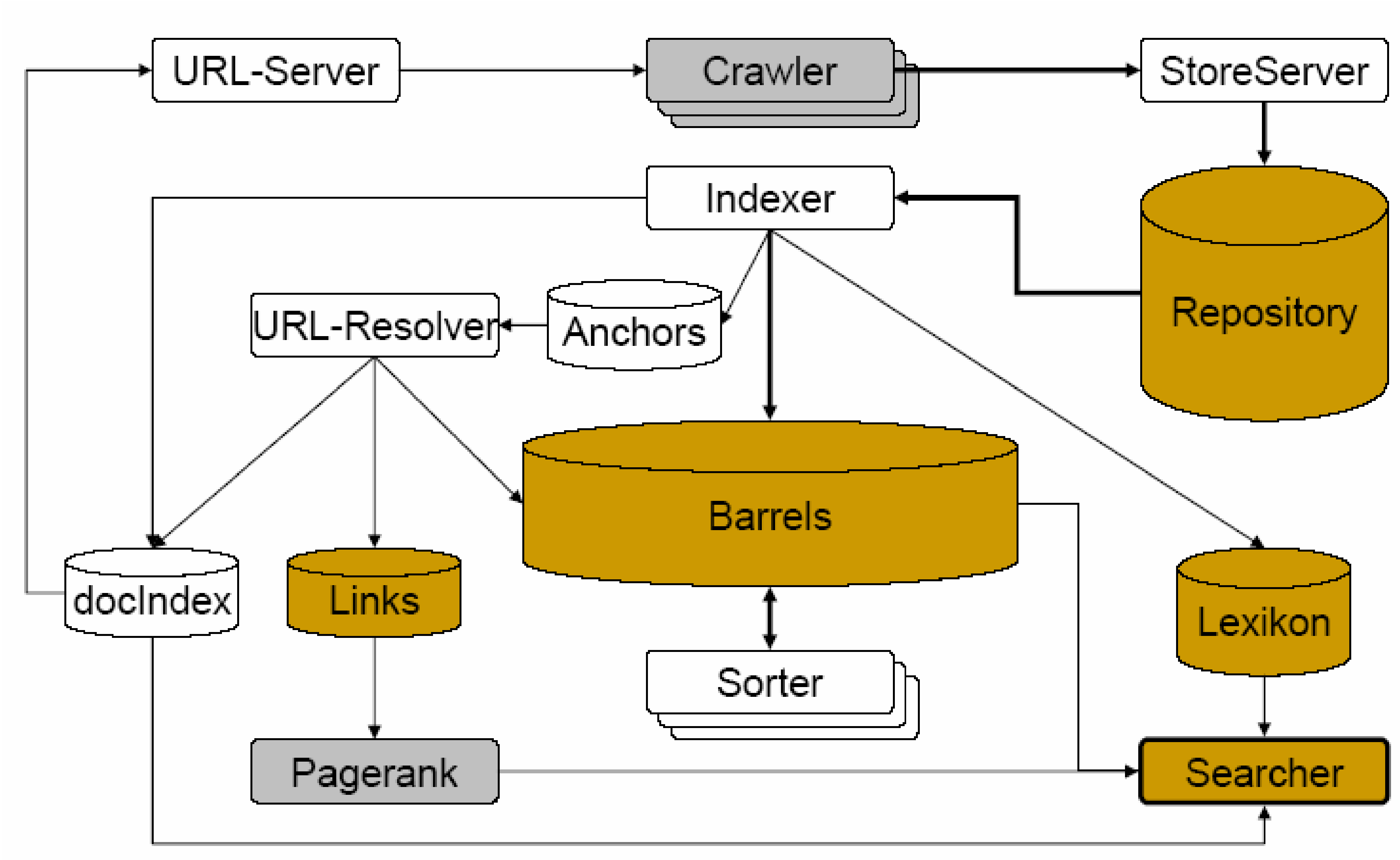
- Classical IR vs. Web IR
  - Classical IR
    - Evaluate by three lines
      - recall、precision 、precision at the top 10 result pages
  - Web IR
    - Relevant is not enough
    - Goal is to return
      - High-relevance
      - High-quality (valuable)

Current Status of Search Engines

- Google
  - Innovative ranking algorithm (more than others)
- AltaVista
  - The largest data collection
- Northern Light
  - Better serving on academic and business topic
- Infoseek
  - Powerful Sub-serach
- FastSearch
  - Second largest data collection

Architecture of A Search Engine

Architecture



Architecture of A Search Engine

Architecture

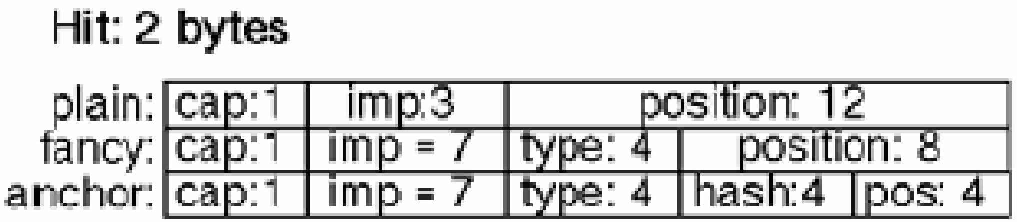
- The web crawler
  - URLserver
  - Storesserver
- The indexer
  - Read & Uncompress docs from Respository
  - Anchor
  - URLresolver
    - Doc Index
    - Barrels
    - Links
  - Sorter
  - DumpLexicon
- The query server
  - Use the lexicon with the inverted index and the PageRanks to answer queries

Architecture of A Search Engine
Data Structure(1/4)

- Repository
  - Contains the full HTML text
  - Compressed using zlib (RFC1950)
  - Prefixed by docID, length, and URL
- Document Index
  - Each entry contain
    - The current doc status (crawled ?)
    - A pointer into the repository (if crawled)
    - A document checksum (using binary search to find the docID)
    - Various statistics
- Lexicon
  - Keep in memory on a 256M
  - Current contains 14 million words

Architecture of A Search Engine
Data Structure(2/4)

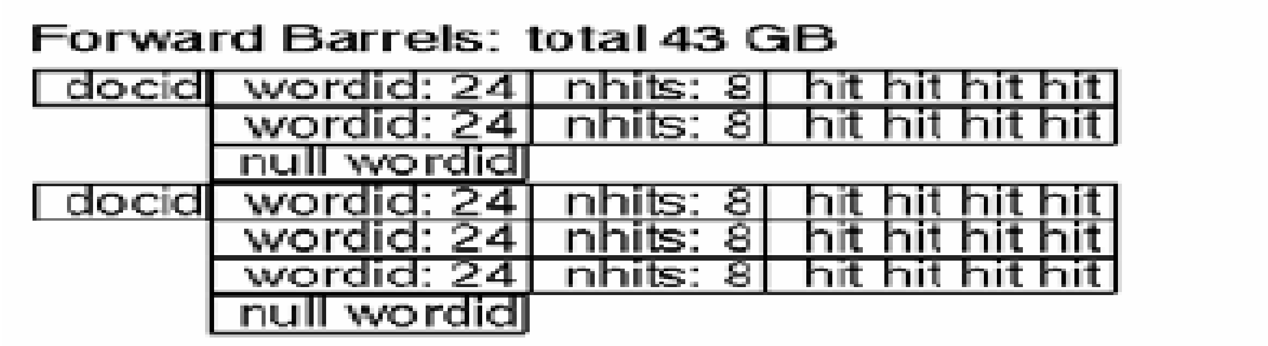
- Hit Lists
  - Encoding by a hand optimized compact



  - Two type (plain hit and fancy hit [imp=111])
  - For anchor hit
    - 4 bits for a hash of the docId (limit for phrase searching)
    - 4 bits for position in anchor

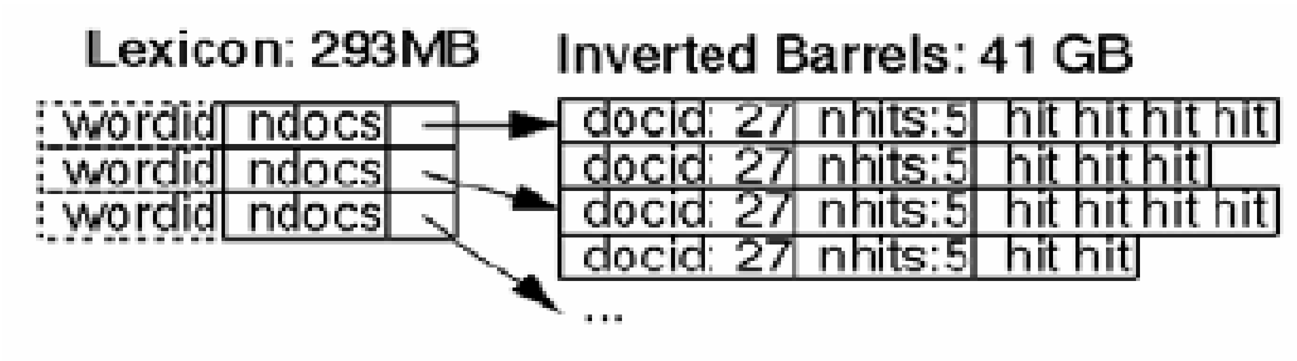
Architecture of A Search Engine
Data Structure(3/4)

- Forward Index
  - 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)

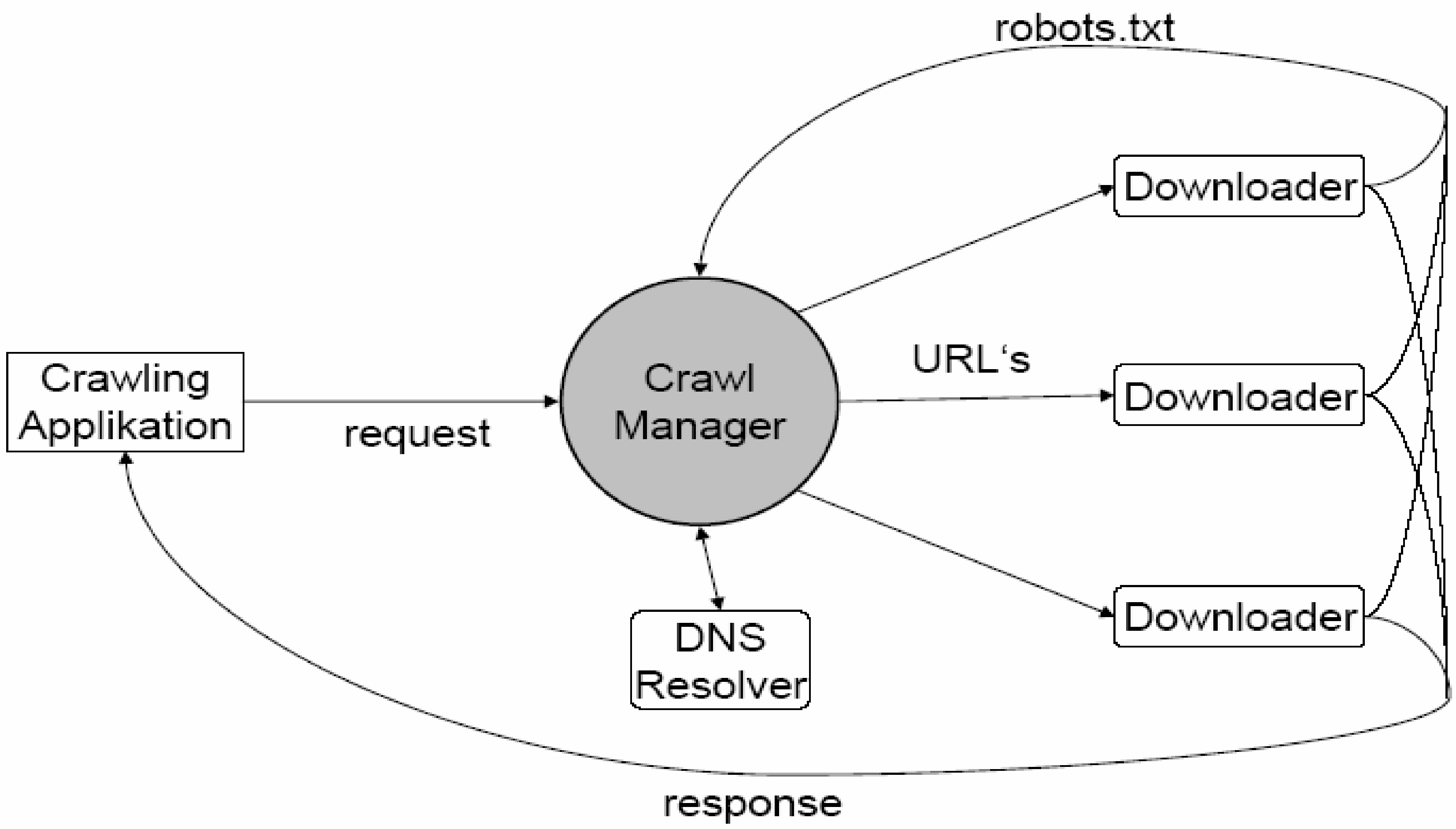
- Inverted Index
  - Importance issue：what order of the doclist
    - Sorted by docID (quick merging the doclists)
    - Sorted by a ranking of the occurrence of the word in each doc
    - Google chose a compromise (keep two sets)
      - One set for hit lists which include title or anchor hits (considered High ranking , first check if there are not enough matches, check another)
      - Another set for all hit lists



Engineering Issues
Crawling the Web (1/2)

- Google crawler
  - Maintain its own DNS cache
  - Asynchronous I/O to manage events
  - 4 crawler
    - Both URLserver & crawler are implement in Python
    - Each crawler keeps 300 connections open at once
    - >100 pages / s , roughly 600K/s
- Cho etc.99 (spread the workload)
  - Allocation that URL's in 500 Queues
  - Allocation based on the Hash of the server name
  - Read one URL from each queue at a time

Engineering Issues
Crawling the Web (2/2)



Engineering Issues
Caching Query Results

- Cache proxies
- Markatos99
  - locality in the queries submitted (20%~30%)
  - Two-stage LRU (LRU-2S) cache replacement
  - Account both recency(LRU) and frequency (LRU-2S)
  - Experiment show that medium-size (a few hundred Mbytes large) caches can result in hit rate

	primary		secondary	
Initial state of the Queue:	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)

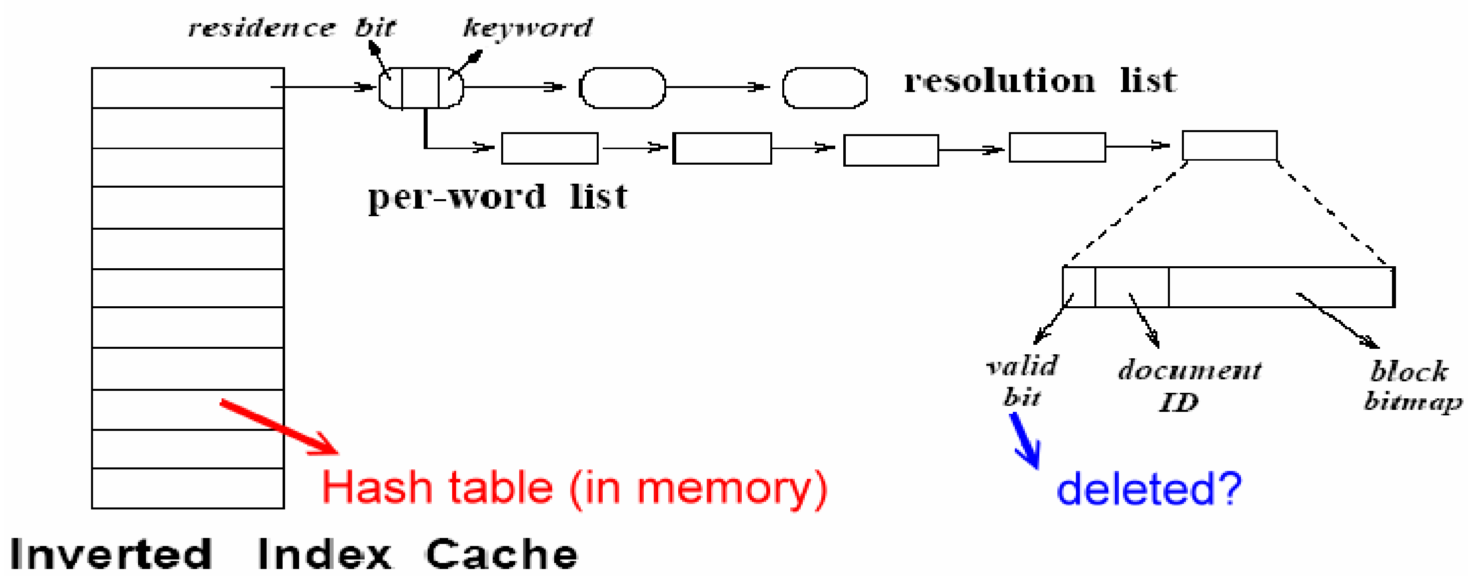
- Callan94 (INQUERY system)
  - Using the Mneme (Moss90) persistent object store to manage its inverted file index
  - When exceed, additional large object are allocated (copy & free old) and chained together in a linked list
  - Lists are allocated using a range of fixed size objects (range from 16 to 8192 bytes by power of 2)
  - Superior perfomance in terms of both time and space , with only a small impact on query processing

Engineering Issues
Incremental Updates to Inverted Index (2/5)

- 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)

- The above solutions
  - keep a second copy (with update operation)
  - Can't update & search simultaneously
- Codir (Author's system L.Huang 98)



Engineering Issues
Incremental Updates to Inverted Index (4/5)

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

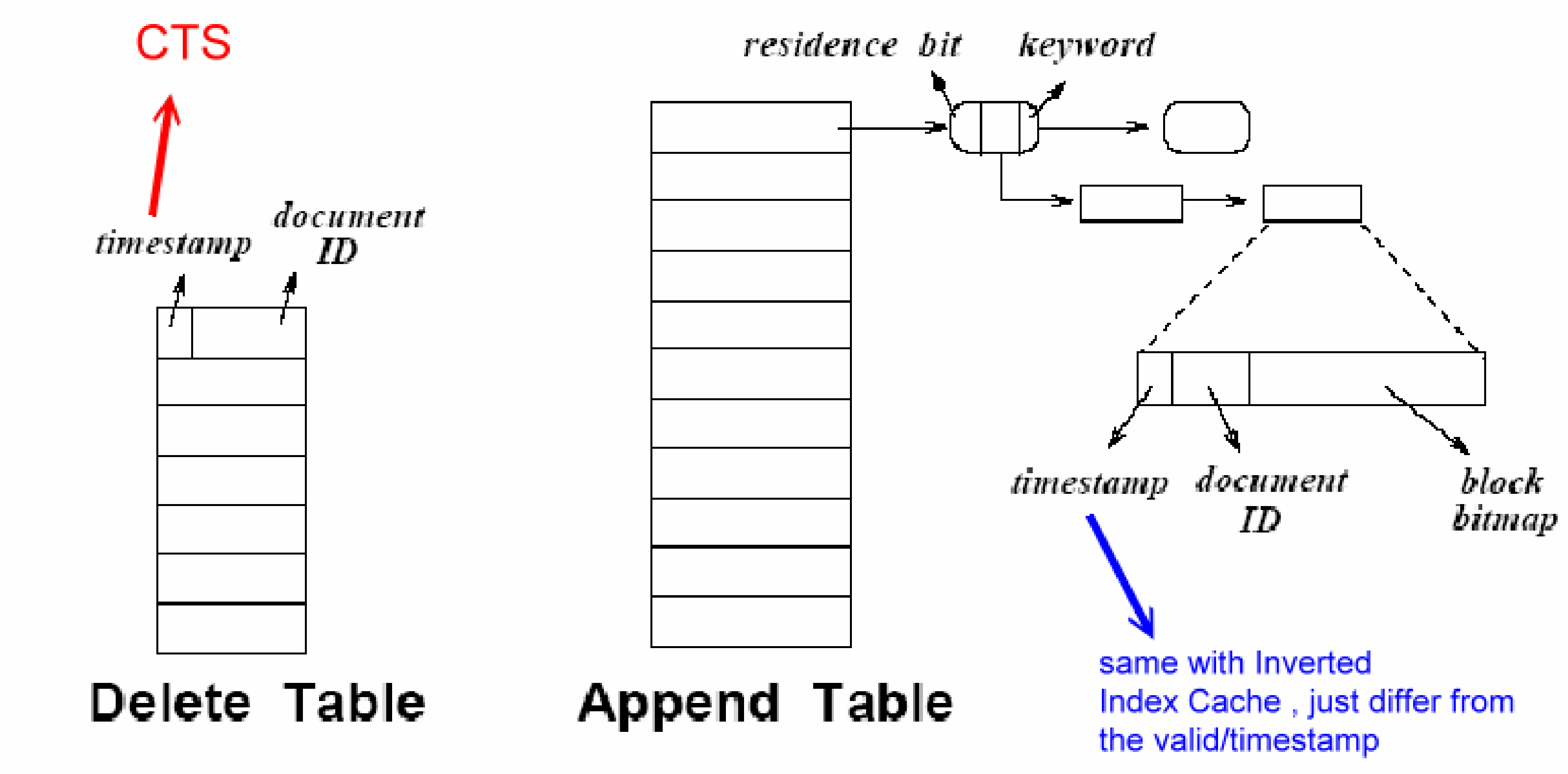


Figure 3: Data Structure Used in Codir

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

- Notation
  - A has pages T1… Tn (citations)
  - d range from 0~1 (google set 0.85)
  - C(A)：number of links going out of page A

$$PR(A) = (1 - d) + d(PR(T1)/C(T1) + ... + PR(Tn)/C(Tn))$$

- The probability that the random surfer visits a page is its PageLink (the d factor)
- High PageRank
  - Many pages pointing to it
  - Or there are some pages that point to it and have a high PageLink

Algorithmic Issues
Ranking- HITS Algorithm(1/19)

- Given a query , HITS will find
  - Authorities
    - good sources of content
    - Large in-degree
  - Hub
    - good sources of links
    - Pull together authorities on a given topic (Like Yahoo)

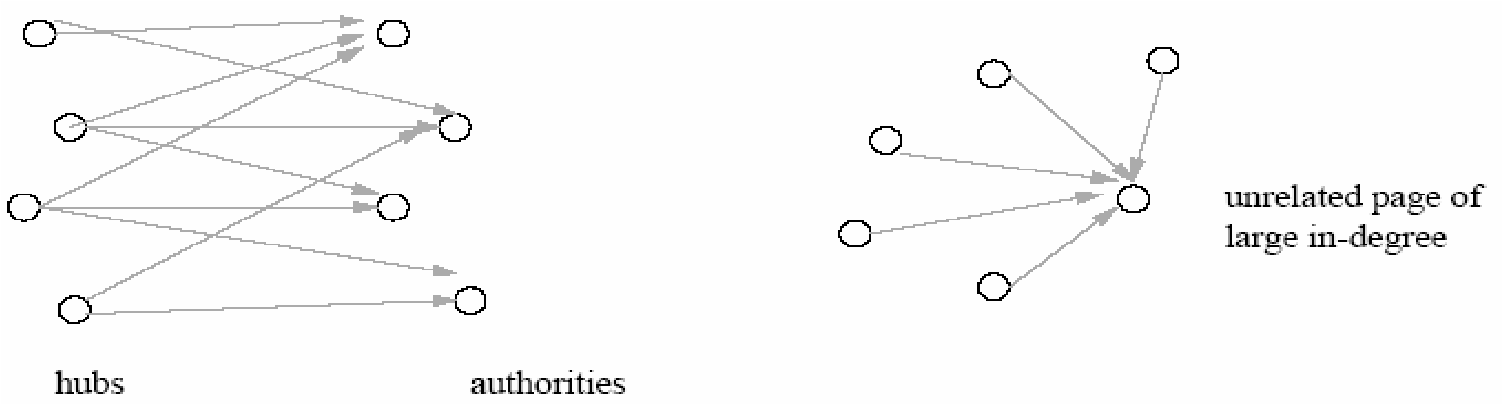


Figure 4: A densely linked set of hubs and authorities

Algorithmic Issues
Ranking- HITS Algorithm(2/19)

- Considering the Web structure
  - page = node
  - link = directed edge
- Links – latent human judgment
- Focused Subgraph
  - 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(3/19)

- Algorithm Overview
  - Input: σ – a query string

    Σ – a text-based search engine

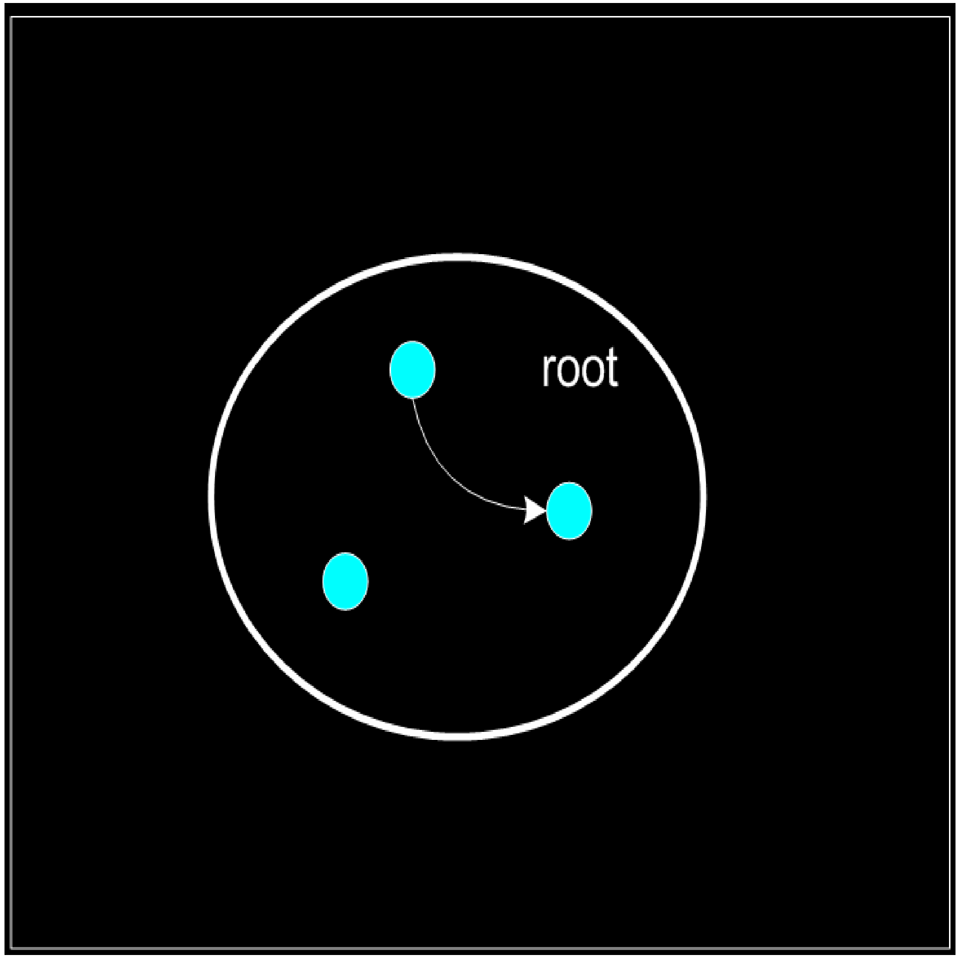
    `t` – size of the *root set*

    d – max number of "in" links

    Top `t` pages (highest-ranked pages) from the text-based search engine form the *root set* (R_σ)
  - Output: $\boldsymbol{S}_\sigma$ – focused subset

Algorithmic Issues
Ranking- HITS Algorithm(4/19)

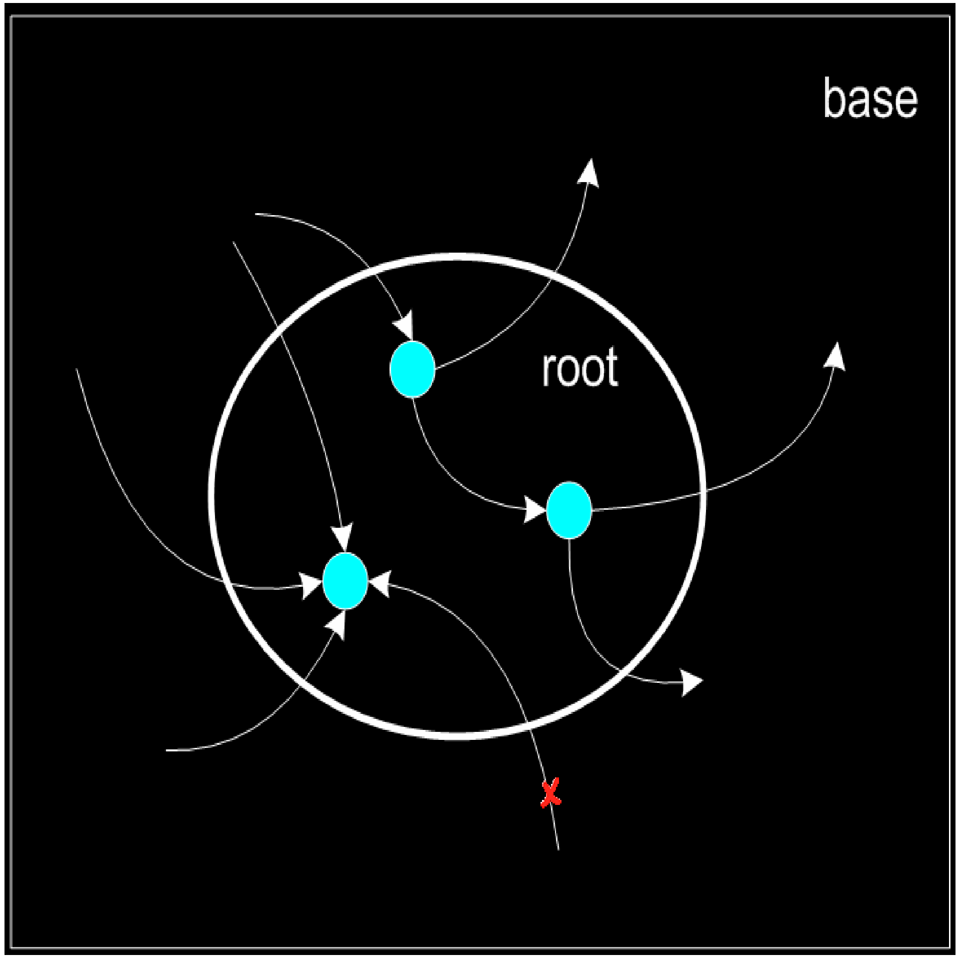
(σ = "java", Σ = AltaVista, t = 3, d = 3)



S_σ

Algorithmic Issues
Ranking- HITS Algorithm(5/19)

(σ = "java", Σ = AltaVista, **t** = 3, **d** = 3)

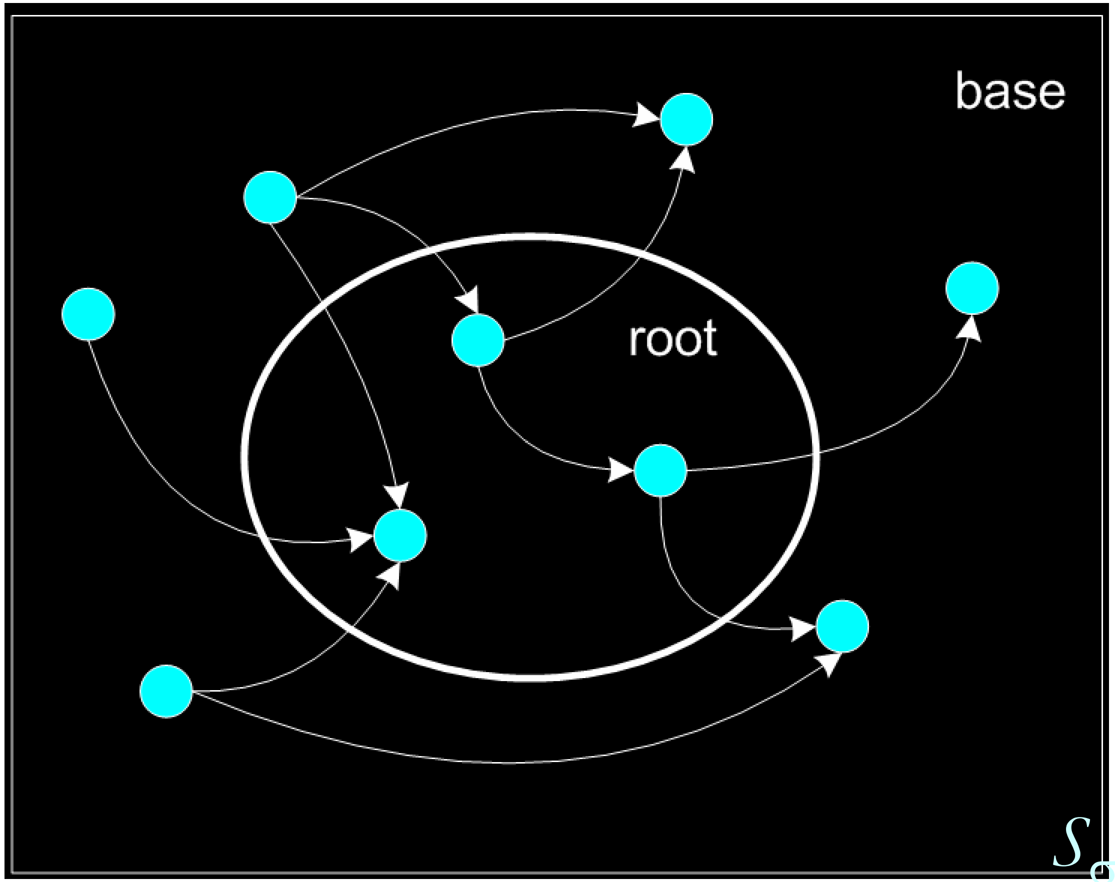


S_σ

Algorithmic Issues

Ranking- HITS Algorithm(6/19)

(σ = "java", Σ = AltaVista, t = 3, d = 3)



Algorithmic Issues
Ranking- HITS Algorithm(7/19)

- An Iterative Algorithm

[authority] weights vector $x_0 = (1, 1, 1, \ldots, 1)$

[hub] weights vector $y_0 = (1, 1, 1, \ldots, 1)$

for $i = 1, 2, \ldots, k$

    x_i = update_authorityw(y_{i-1})

    y_i = update_hubw(x_i)

    normalize(x_i, y_i)

return (x_k, y_k)

Algorithmic Issues

Ranking- HITS Algorithm(8/19)

$x_0 = (1, 1, 1, \ldots, 1)$

$y_0 = (1, 1, 1, \ldots, 1)$

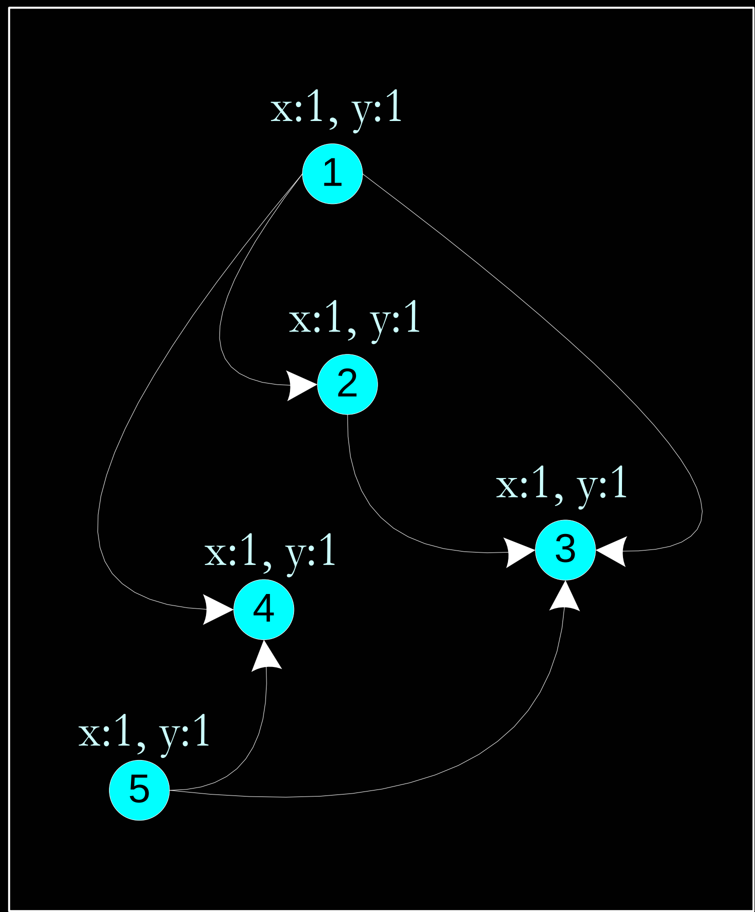
for $i = 1, 2, \ldots, k$

  $x_i = \text{update\_auth}(y_{i-1})$

  $y_i = \text{update\_hub}(x_i)$

  normalize(x_i, y_i)

return (x_k, y_k)



Algorithmic Issues
Ranking- HITS Algorithm(9/19)

$x_0 = (1, 1, 1, \ldots, 1)$

$y_0 = (1, 1, 1, \ldots, 1)$

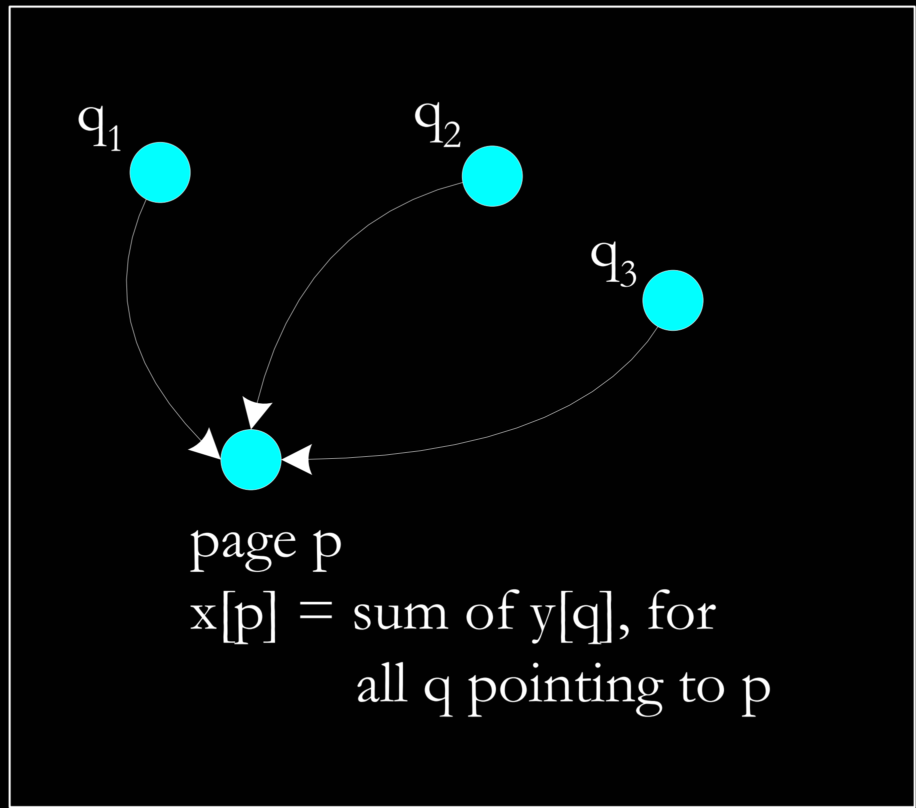
for $i = 1, 2, \ldots, k$

  $x_i = \text{update\_auth}(y_{i-1})$

  $y_i = \text{update\_hub}(x_i)$

  $\text{normalize}(x_i, y_i)$

return (x_k, y_k)



authority weight – x

Algorithmic Issues

Ranking- HITS Algorithm(10/19)

$x_0 = (1, 1, 1, \ldots, 1)$

$y_0 = (1, 1, 1, \ldots, 1)$

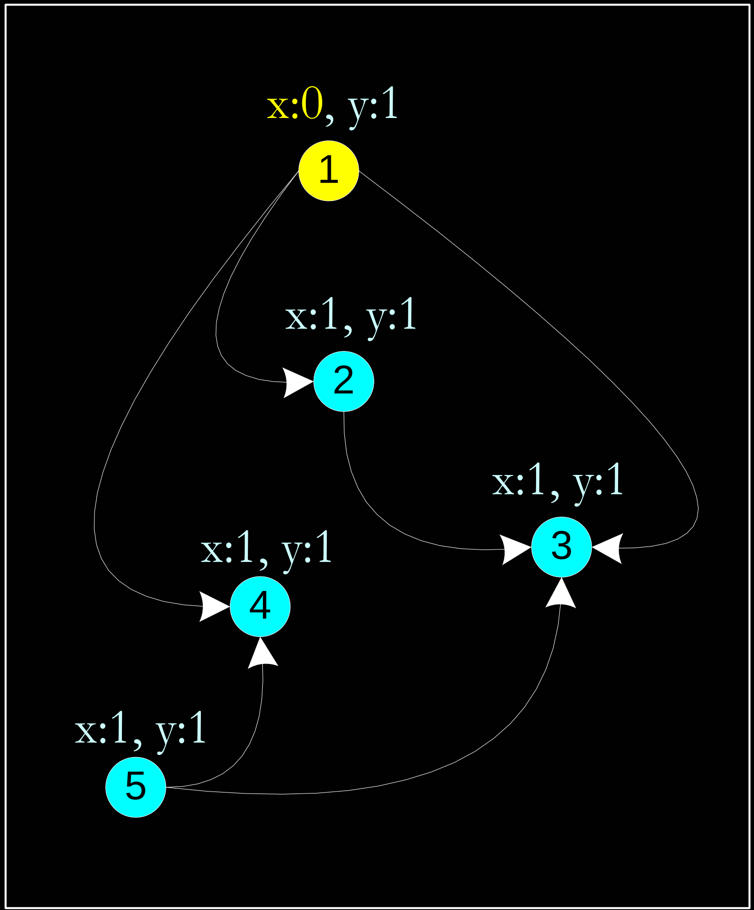
for $i = 1, 2, \ldots, k$

  $x_i = \text{update\_auth}(y_{i-1})$

  $y_i = \text{update\_hub}(x_i)$

  $\text{normalize}(x_i, y_i)$

return (x_k, y_k)



Algorithmic Issues

Ranking- HITS Algorithm(11/19)

$x_0 = (1, 1, 1, \ldots, 1)$

$y_0 = (1, 1, 1, \ldots, 1)$

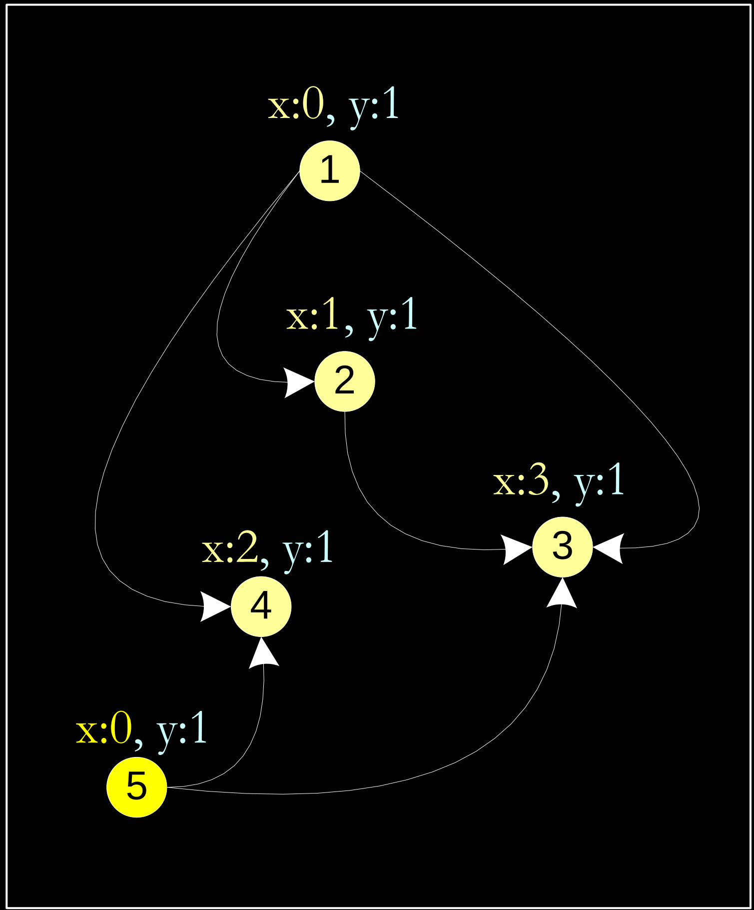
for $i = 1, 2, \ldots, k$

  $x_i = \text{update\_auth}(y_{i-1})$

  $y_i = \text{update\_hub}(x_i)$

  $\text{normalize}(x_i, y_i)$

return (x_k, y_k)



Algorithmic Issues
Ranking- HITS Algorithm(12/19)

$x_0 = (1, 1, 1, \ldots, 1)$

$y_0 = (1, 1, 1, \ldots, 1)$

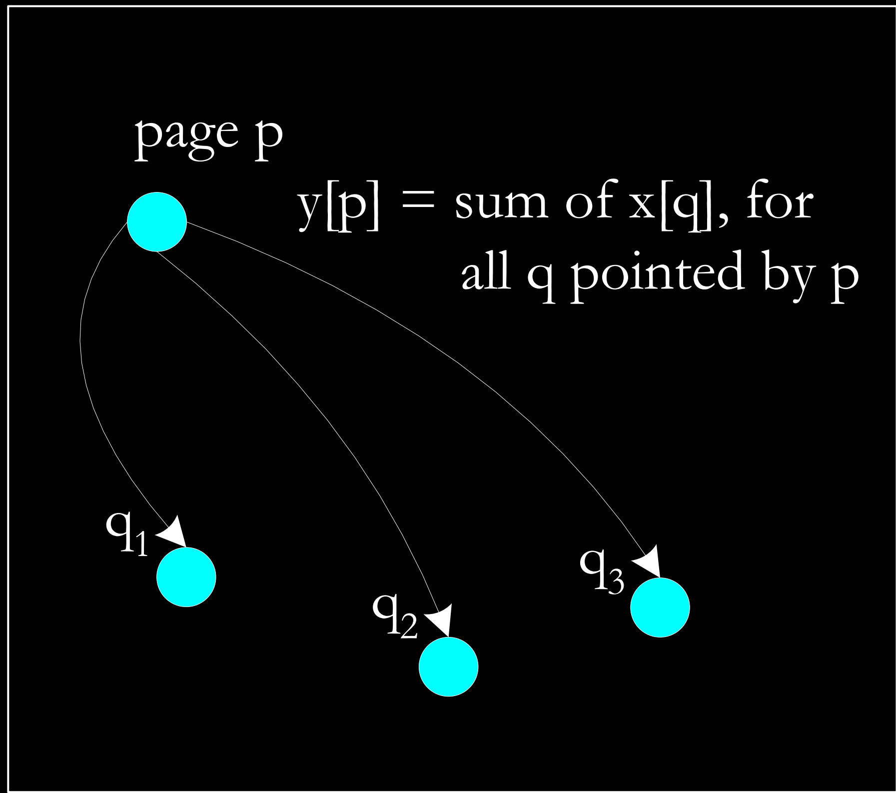
for $i = 1, 2, \ldots, k$

   x_i = update_auth(y_{i-1})

   y_i = update_hub(x_i)

   normalize(x_i, y_i)

return (x_k, y_k)



hub weight – y

Algorithmic Issues
Ranking- HITS Algorithm(13/19)

$x_0 = (1, 1, 1, \ldots, 1)$

$y_0 = (1, 1, 1, \ldots, 1)$

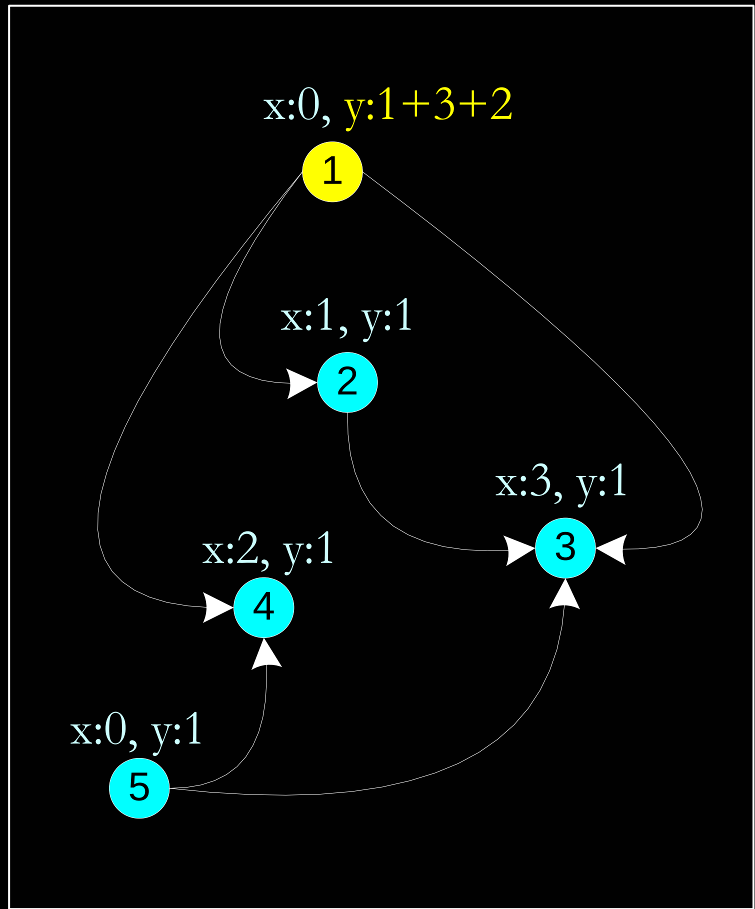
for $i = 1, 2, \ldots, k$

  x_i = update_auth(y_{i-1})

  y_i = update_hub(x_i)

  normalize(x_i, y_i)

return (x_k, y_k)



Algorithmic Issues

Ranking- HITS Algorithm(14/19)

$x_0 = (1, 1, 1, \ldots, 1)$

$y_0 = (1, 1, 1, \ldots, 1)$

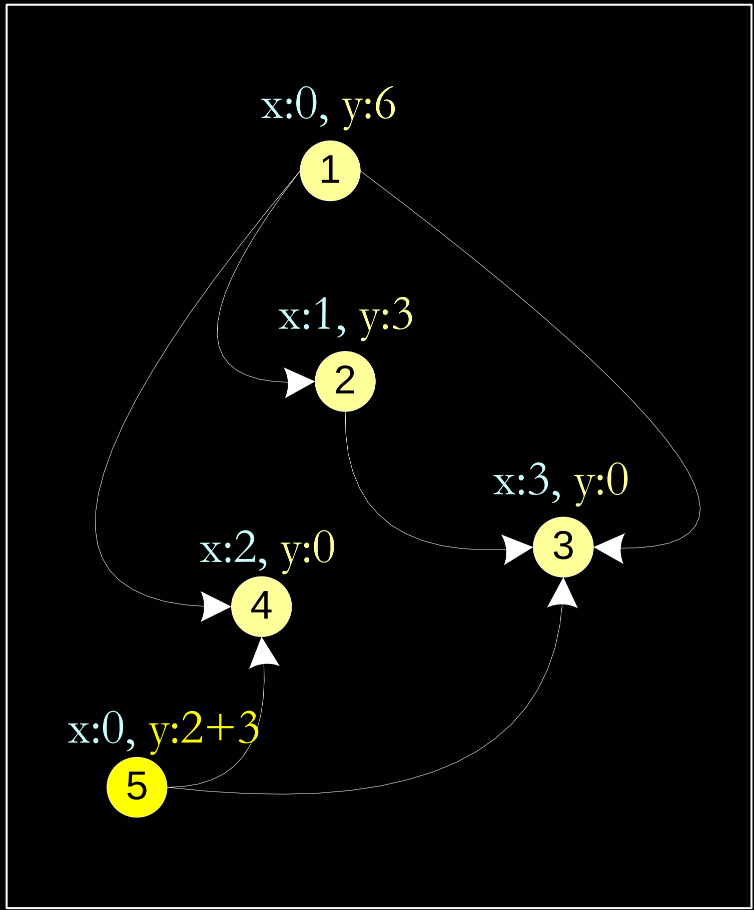
for $i = 1, 2, \ldots, k$

  $x_i = \text{update\_auth}(y_{i-1})$

  $y_i = \text{update\_hub}(x_i)$

  $\text{normalize}(x_i, y_i)$

return (x_k, y_k)



Algorithmic Issues

Ranking- HITS Algorithm(15/19)

$x_0 = (1, 1, 1, \ldots, 1)$

$y_0 = (1, 1, 1, \ldots, 1)$

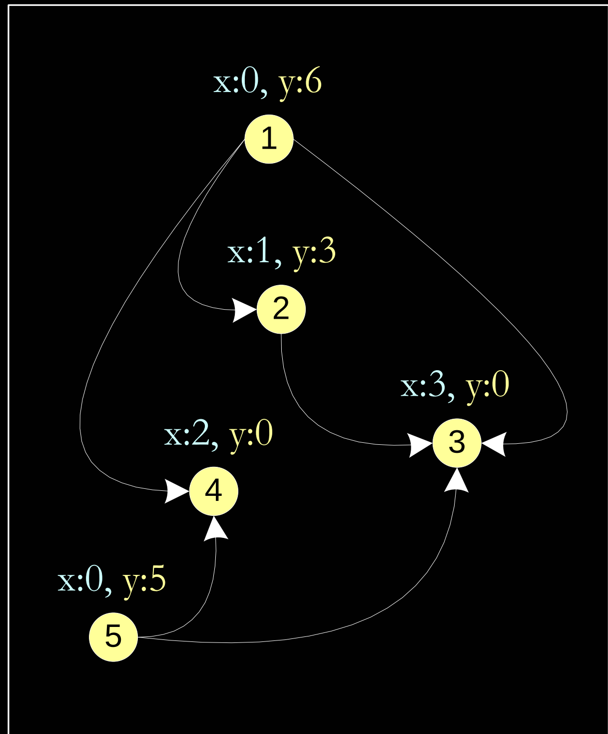
for $i = 1, 2, \ldots, k$

$x_i = \text{update\_auth}(y_{i-1})$

$y_i = \text{update\_hub}(x_i)$

$\text{normalize}(x_i, y_i)$

return (x_k, y_k)



$\text{normalize}(x_i)$

$0+1+3+2+0 = 6$

Algorithmic Issues
Ranking- HITS Algorithm(16/19)

$x_0 = (1, 1, 1, \ldots, 1)$

$y_0 = (1, 1, 1, \ldots, 1)$

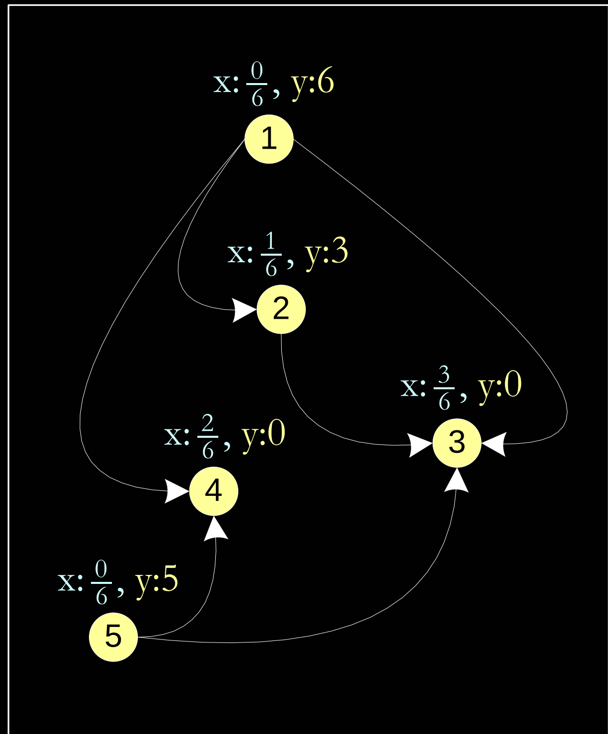
for $i = 1, 2, \ldots, k$

  $x_i = \text{update\_auth}(y_{i-1})$

  $y_i = \text{update\_hub}(x_i)$

  $\text{normalize}(x_i, y_i)$

return (x_k, y_k)



$\text{normalize}(y_i)$

$6+3+0+0+5 = 14$

Algorithmic Issues

Ranking- HITS Algorithm(17/19)

$x_0 = (1, 1, 1, \ldots, 1)$

$y_0 = (1, 1, 1, \ldots, 1)$

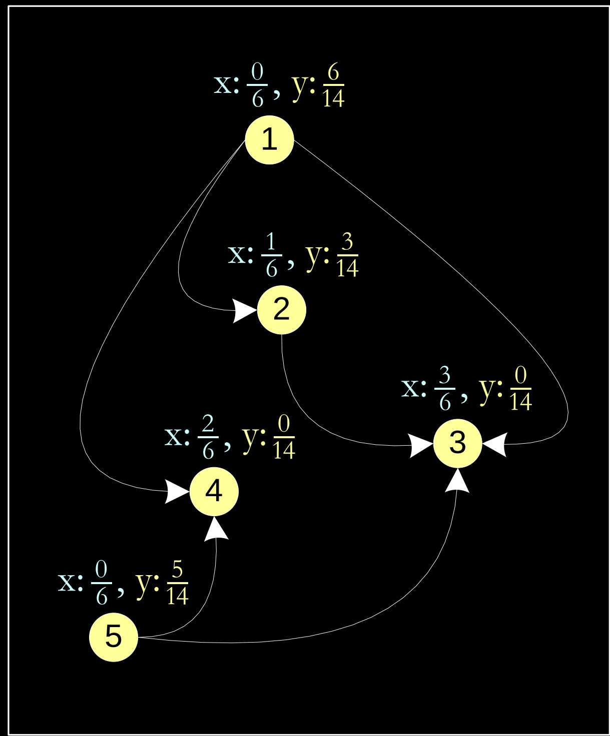
for $i = 1, 2, \ldots, k$

  $x_i = \text{update\_auth}(y_{i-1})$

  $y_i = \text{update\_hub}(x_i)$

  $\text{normalize}(x_i, y_i)$

return (x_k, y_k)



$\text{normalize}(y_i)$

$6+3+0+0+5 = 14$

Algorithmic Issues

Ranking- HITS Algorithm(18/19)

$x_0 = (1, 1, 1, \ldots, 1)$

$y_0 = (1, 1, 1, \ldots, 1)$

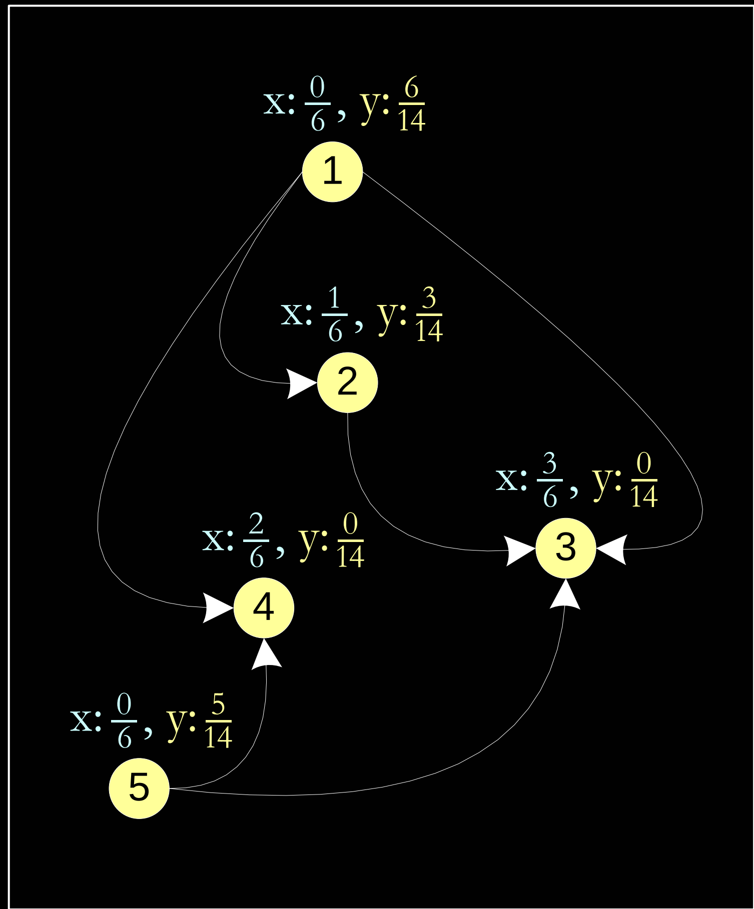
for $i = 1, 2, \ldots, k$

$\quad x_i = \text{update\_auth}(y_{i-1})$

$\quad y_i = \text{update\_hub}(x_i)$

$\quad \text{normalize}(x_i, y_i)$

return (x_k, y_k)



Algorithmic Issues
Ranking- HITS Algorithm(19/19)

- HITS didn’t work well
  - Mutually Reinforcing Relationship Between Hosts
  - Automatically Generated Links
  - Non-Relevant Node
- Bharat 98
  - Topic drift approach
  - K edges➔1/k authority weight
  - L edges➔1/l hub weight

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

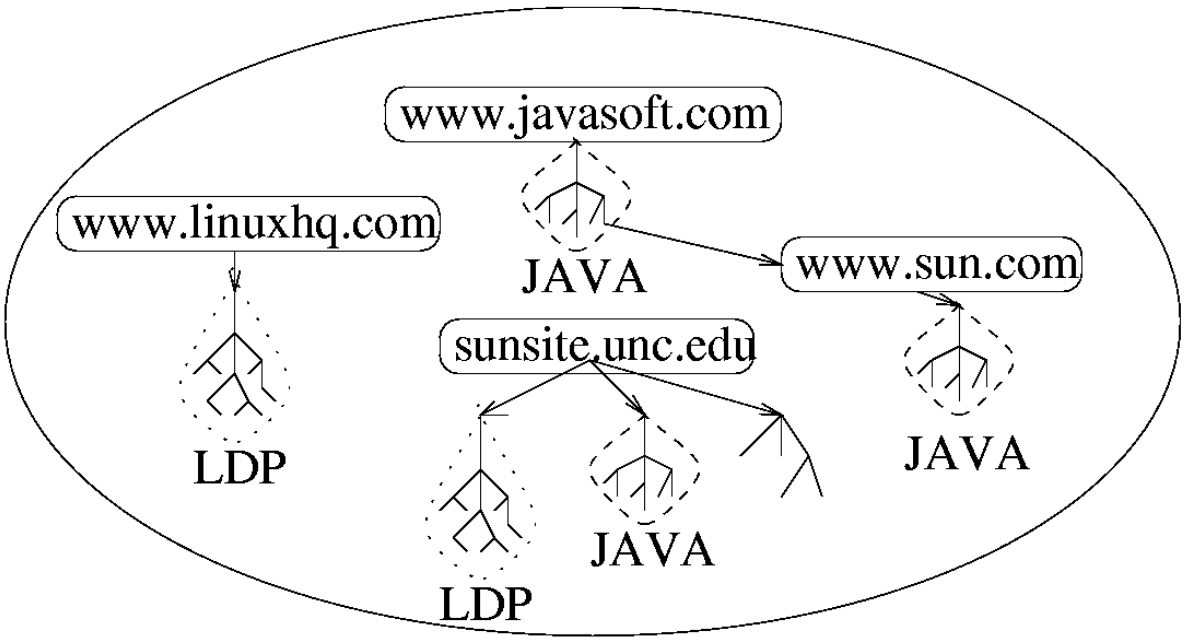
$$H[n] := \sum_{(n',n)\in N} A[n'] \times hub\_wt(n',n)$$

Algorithmic Issues
Ranking- Others

- Anchor text advantage
  - Provide more accurate descriptions of web pages
  - Deal with docs that can’t be indexed (ex: image)
- Cutler97
  - Assign different weight to heading as well as anchor text (help WebIR)
  - Conclusion is that anchor texts and STRONG class should carry more weight(STRONG、B、OL、UL)

Algorithmic Issues
Duplicate Elimination(1/13)

- Quote <Junghoo Cho 99>
  - Approximately 30% of pages are (near) duplicates!

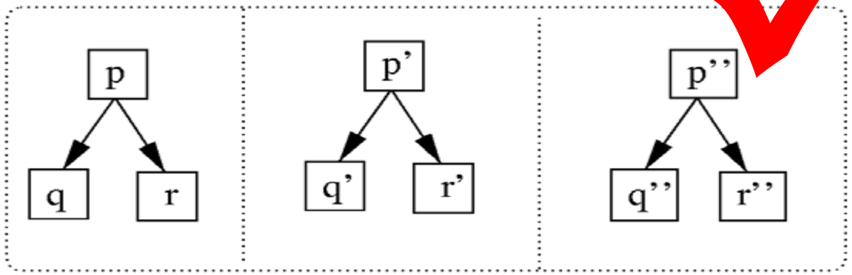


Algorithmic Issues
Duplicate Elimination(2/13)

- 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
  - Subgraph isomorphism: NP

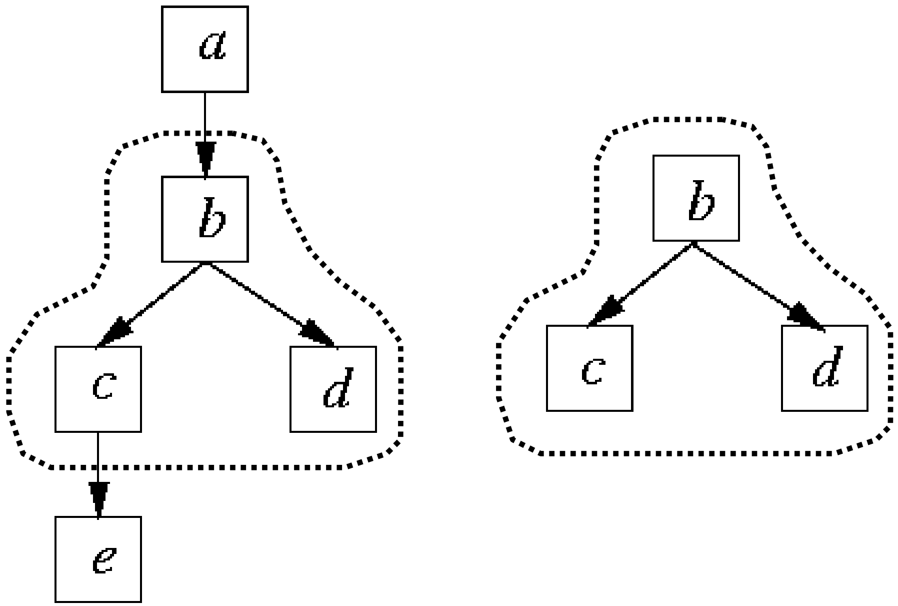
Algorithmic Issues
Duplicate Elimination(3/13)

- Page content similarity
  - Fingerprint-based(32bit) approach (chunking)
    - Shingles [Broders et al., 1997]
    - Sentence [Brin et al., 1995]
    - Word [Shivakumar et al., 1995]
  - Interesting issues
    - Threshold value T
  - Transitive similary



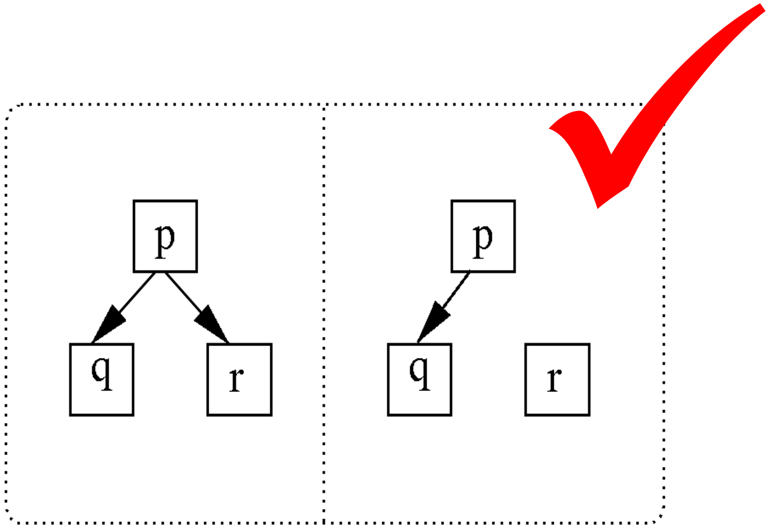
Algorithmic Issues
Duplicate Elimination(4/13)

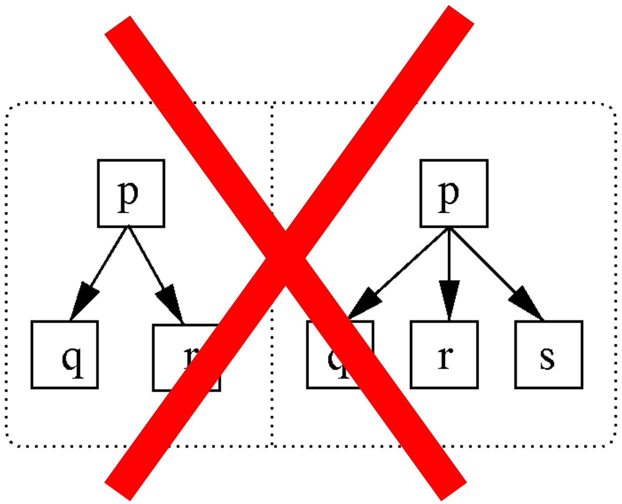
- Identical Collection
  - Collection: induced subgraph
  - one-to-one mapping
    - Identical pages
    - Identical link structure



Algorithmic Issues
Duplicate Elimination(5/13)

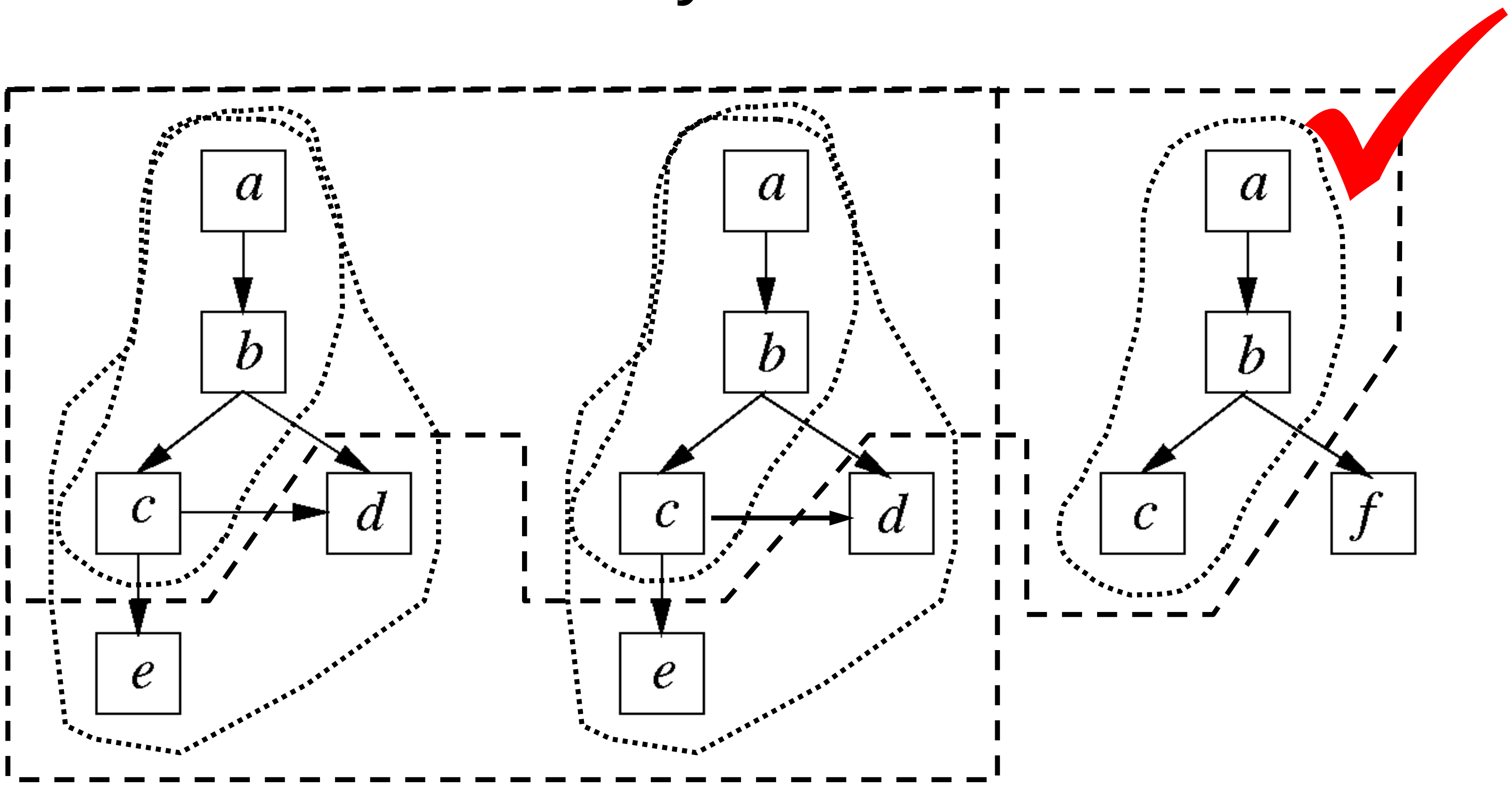
- Similar Collection
  - one-to-one mapping
    - similar pages
    - similar link structure
  - Size (equi-sized collection must identify)





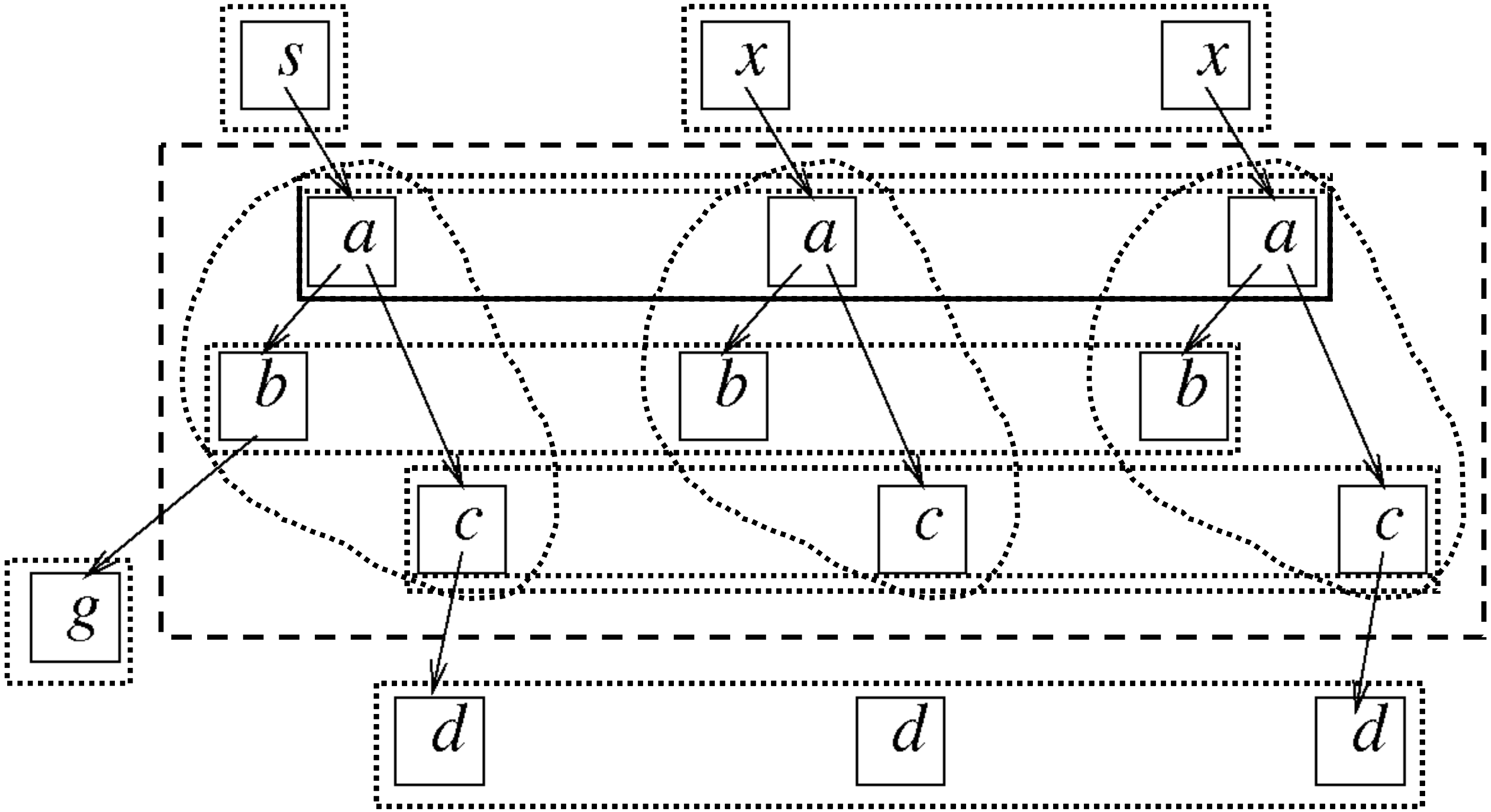
Algorithmic Issues
Duplicate Elimination(6/13)

- Size vs. Cardinality



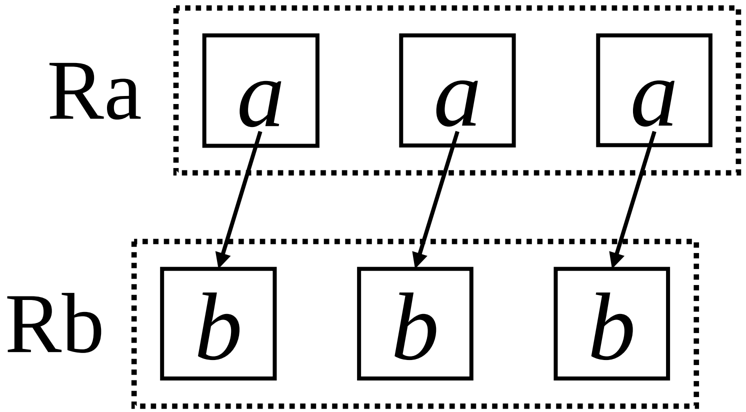
Algorithmic Issues
Duplicate Elimination(7/13)

- Growth strategy



Algorithmic Issues
Duplicate Elimination(8/13)

- Essential property (Merge condition)



Ls: # of pages linked *from*

Ld: # of pages linked *to*

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

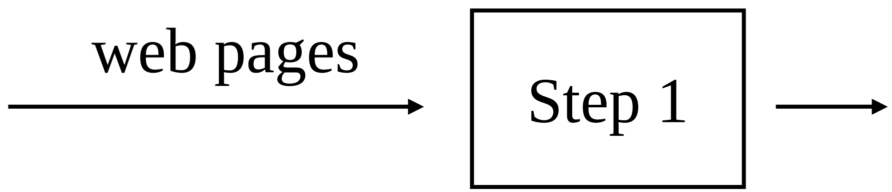
Algorithmic Issues
Duplicate Elimination(9/13)

- Algorithm
  - Based on the property we identified
  - Input: set of pages collected from web
  - Output: set of similar collections
  - Complexity: O(n log n)

Algorithmic Issues

Duplicate Elimination(10/13)

- Step 1: Similar page identification (iceberg query DSGM98)



Rid	Pid
1	10375
1	38950
1	14545
2	1026
2	18633

25 million pages

Fingerprint computation: 44 hours

Replicated page computation: 10 hours

Algorithmic Issues
Duplicate Elimination(11/13)

- Step 2: link structure check

R1

Rid	Pid
1	10375
1	38950
1	14545
2	1026

$\bowtie$

Link

Pid	Pid
1	2
1	3
2	6
2	10

$\bowtie$

R2 (Copy of R1)

Rid	Pid
1	10375
1	38950
1	14545
2	1026

$\Downarrow$ Group by (R1.Rid, R2.Rid)

Ra = |R1|, Ls = Count(R1.Rid), Ld = Count(R2.Rid), Rb = |R2|

Algorithmic Issues
Duplicate Elimination(12/13)

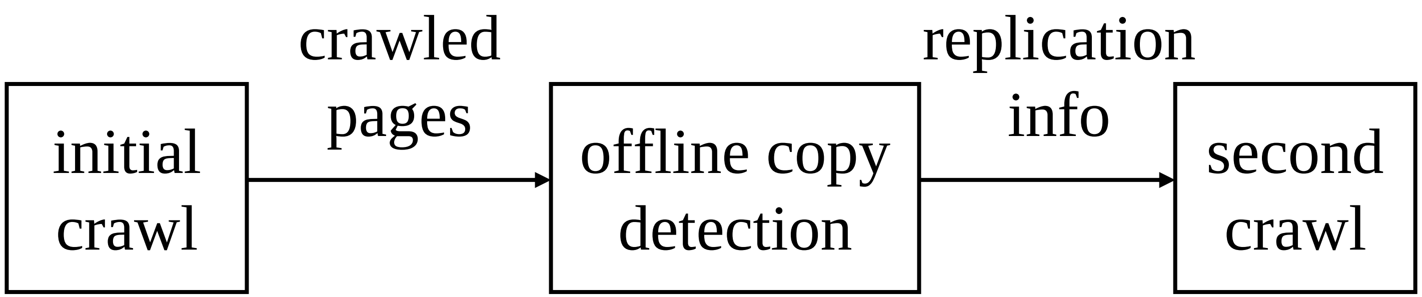
- Step 3:

```
S = {}
For every (|Ra|, Ls, Ld, |Rb|) in step 2
    If (|Ra| = Ls = Ld = |Rb|)
        S = S U {<Ra, Rb>}
Union-Find(S)    // find connected component
```

- Step 2-3: 10 hours

Algorithmic Issues
Duplicate Elimination(13/13)

- Applications
  - Web crawling & archiving
    - Save network bandwidth
    - Save disk storage



Hierarchical Directories and Automatic Categorization

- Current Status of Hierarchical Directories
- Automatic Categorization 1-Taper
- Automatic Categorization 2-OpenGrid and ODP

Current Status of Hierarchical Directories

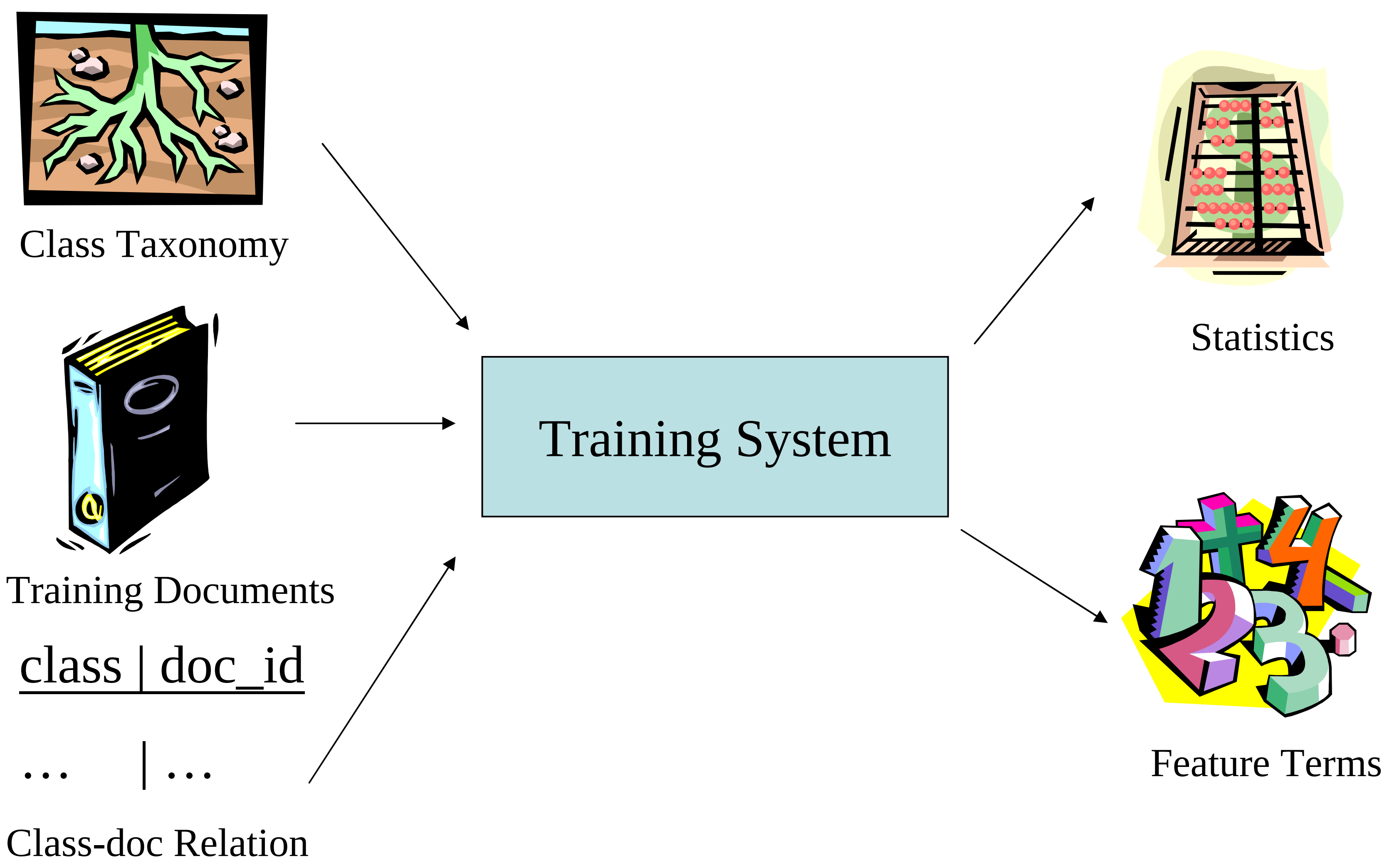
name	Librarians' Index	Infomine	Britannica Web's Best	Yahoo!	Galaxy
Size,type	About 5,000. Compiled by public librarians in information supply business. Highest quality sites only. Great annotations.	About 16,000. Compiled by academic librarians.	About 150,000. Hand-picked, annotated, and ranked by Britannica 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)

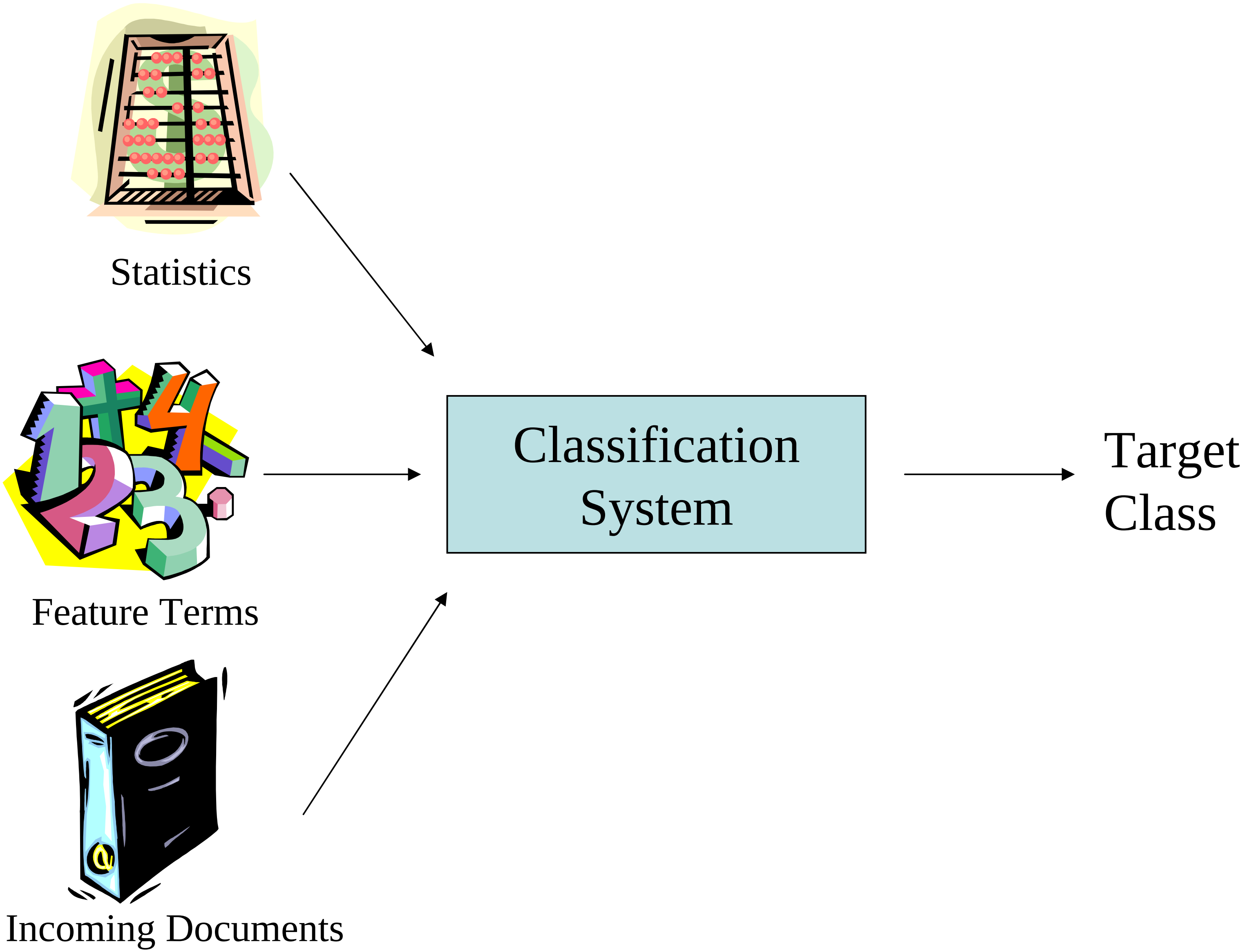
Automatic Categorization 1-Taper(1/16)

- Taper
  - A taxonomy-and-path-enhanced-retrieval system
  - Given
    - Hypertext document corpus
    - A “small” set of classified 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 1-Taper(2/16)



Automatic Categorization 1-Taper(3/16)



Automatic Categorization 1-Taper(4/16)

- Statistics Collection
  - 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

Automatic Categorization 1-Taper(5/16)

- Fisher's discrimination

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

  - c, c_1, c_2 : children of internal node c_0
  - $f(t,d,c)$: the number of times term t occurs in doc d in the training set of class c , with doc length normalized to 1
  - $$\mu(c,t) = \frac{1}{|c|} \sum_{d \in c} f(d,c,t)$$
- Good discriminating power: large interclass distance, small intraclass distance
- Pick the top F terms

Automatic Categorization 1-Taper(6/16)

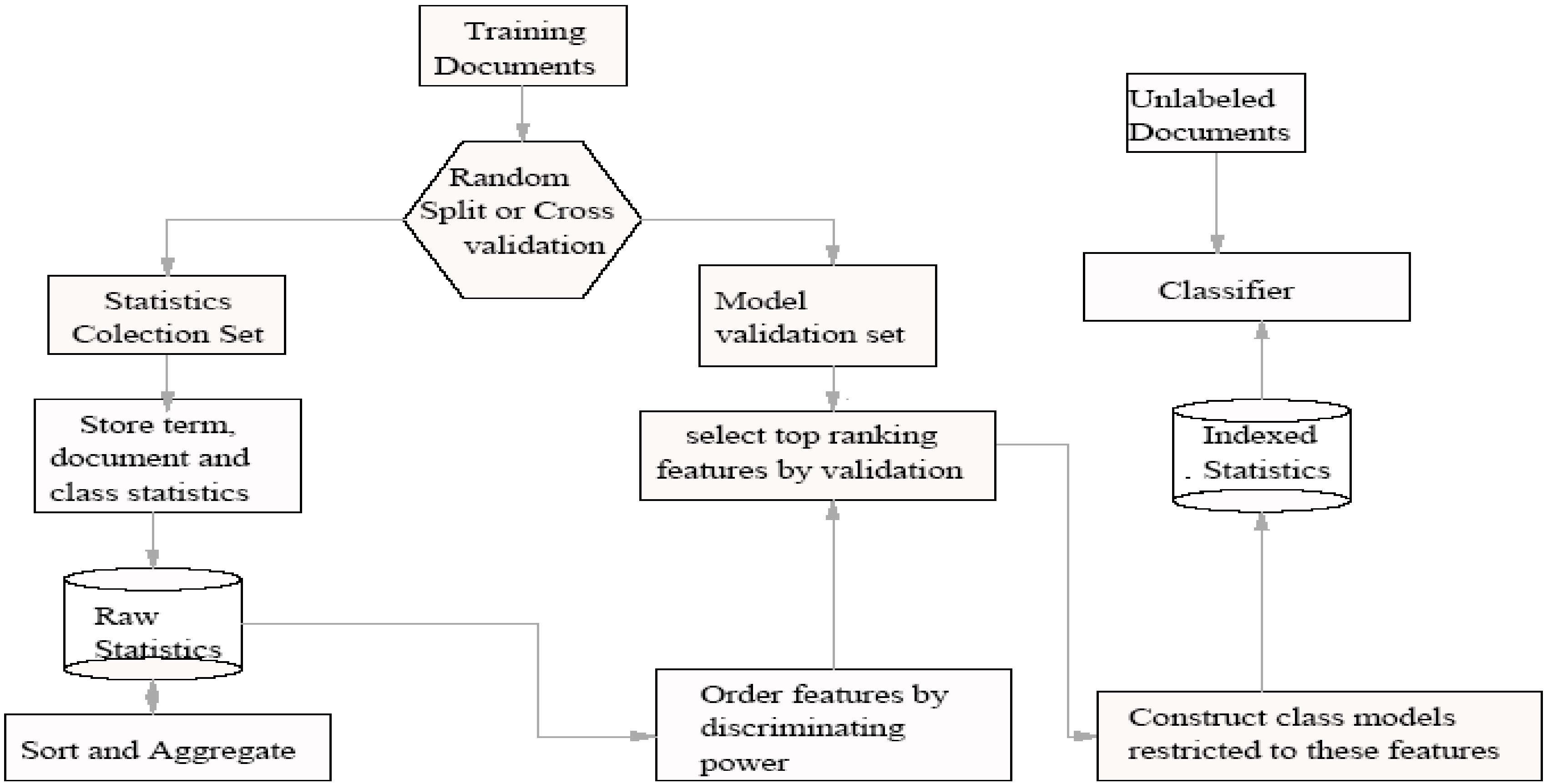


Figure 7: A sketch of the TAPER hierarchical feature selection and classification engine

Automatic Categorization 1-Taper(7/16)

- Evaluation
  - Suppose c_0 has children $c_1, \ldots, c_l$ 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. c_0
  - When a new doc is input , the classifier evaluate the class models and Bayes' law

Automatic Categorization 1-Taper(8/16)

- Evaluation
  - 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 1-Taper(9/16)

- Native Bayes Models (Binary Model)
  - Each parameter indicates the probability that a document in class c will mention term t at least once (classification can pose as a shortest path problem on taxonomy)

$$\Pr(d \mid c) = \prod_{t \in d} \phi_{c,t} \prod_{t \in W, t \notin d} (1-\phi_{c,t}) = \prod_{t \in d} \frac{\phi_{c,t}}{1-\phi_{c,t}} \underbrace{\prod_{t \in W} (1-\phi_{c,t})}_{\text{to account for } d \notin D}$$

- Native Bayes Models (Multinomial model , using it)
  - Each class is modeled with a |term| sided coin.
  - each parameter denotes probability of the face turning up on tossing the die.
  - term t occurs n(d; t) times in document d,
  - document length is a random variable denoted L,

$$\Pr(d \mid c) = \Pr(L = l_d \mid c) \Pr(d \mid l_d, c) = \Pr(L = l_d \mid c) \binom{l_d}{\{n(d,t)\}} \prod_{t \in d} \theta_t^{n(d,t)}$$

Automatic Categorization 1-Taper(10/16)

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

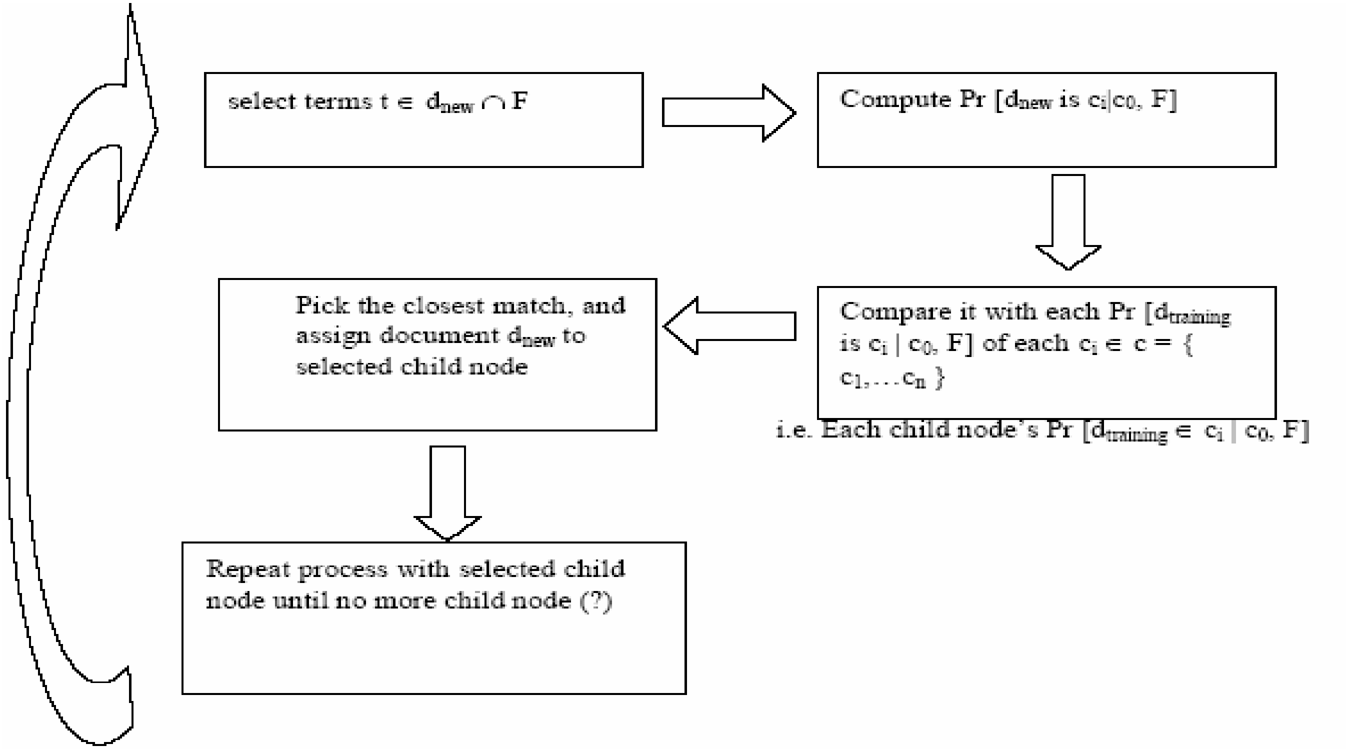
$$\Pr[d \in c \mid c_0, F] = \frac{\text{(prob of d in c) * (prob of t in class c)}^{\text{times t occurred in d}}}{\text{Sum of numerator for all classes c = \{ c}_1\text{,...c}_l\text{ \}}}$$

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

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

Automatic Categorization 1-Taper(11/16)

- Text-only classifiers have Lower accuracy on hyperlinked corpora
  - Heterogenous
  - Information in links not utilized



Automatic Categorization 1-Taper(12/16)

- 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}= {δ_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} = {τ_{ij} , j= 1, 2, ... |di|}

T= {τ_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 1-Taper(13/16)

- Radius-one specialization
  - Bootstrap mechanism
    - 1. Classifying unclassified documents from neighborhood of δ_i using term-only classifier
    - 2. Then, use this information to classify δ_i
    - Iterative 1&2 until constraint
  - **Feature engineering**
    - The core strategy in classification remains the same as before. (Ex：Parent/neighbor)
    - Ex

      If the classes for all documents neighboring δ_i were known, replacing each hyperlink in δ_i with class ID of the corresponding document

Automatic Categorization 1-Taper(14/16)

- Radius-one specialization
  - Choose C to maximize Pr (C|G,T)

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

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

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

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

Automatic Categorization 1-Taper(15/16)

- Pseudocode sketch

  Given test node A

  Construct a radius-r subgraph G around A

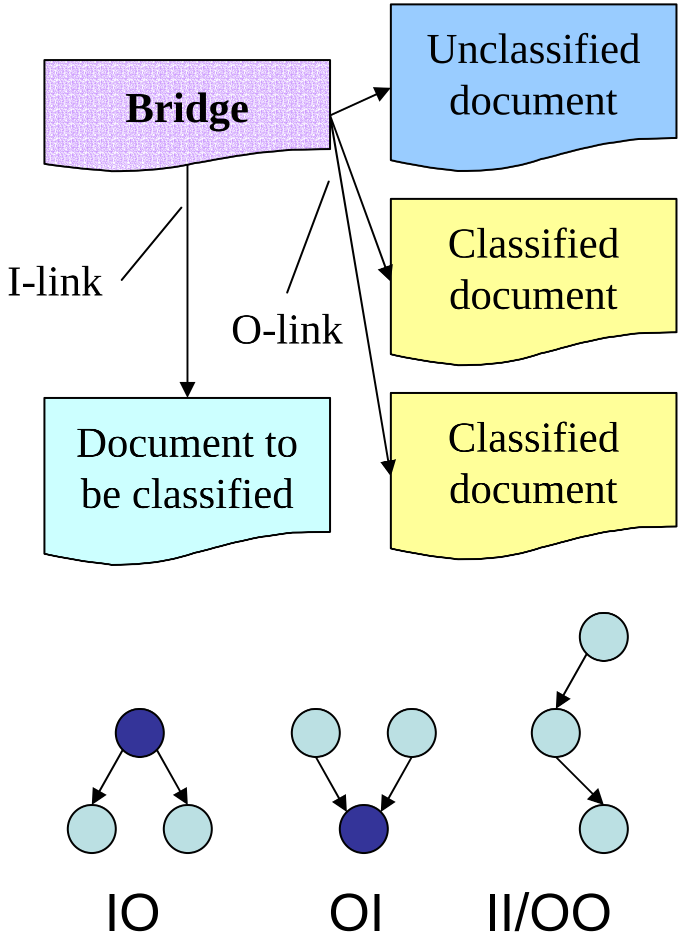
  Assign initial classes to all docs in G using local text

  Iterate until consistent：

      Recompute the class for each doc based on local text and class of neighbors

Automatic Categorization 1-Taper(16/16)

- An "IO-bridge" connects to many pages of similar topics
- "OI" tends to be noisy (many topics point to Netscape and Free Speech Online)
- "II" and "OO" lead to topic divergence



Automatic Categorization 2-OpenGrid and ODP(1/2)

- 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.
  - Centralized system
  - Rank homepages as cool pages and not-so-cool

Automatic Categorization 2-OpenGrid and ODP(2/2)

- OpenGrid system
  - Distributed system utilizing all potential web surfers' opinions and not restricted to number of registered volunteers as OOP.
  - Extension of HTML
    - Classifying field , named cat
    - A field indicating evaluation of the page

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

  - Search all such opinion rank & the voting link to decide the output
  - Still a proposal , no system is running yet.

Measuring the Web(1/14)

- Typical Questions
  - Which search engine has the largest coverage?
  - How many pages are out there and how many are indexed?
- 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(2/14)

- Procedures for Implementation
  - **Sampling**: A procedure for picking pages uniformly at random from the index of a particular engine
  - **Checking**: A procedure for determining whether a particular page is indexed by a particular engine
  - Problem: you need privileged access to a search engine's database
  - Solution: construct good approximations that use only queries

Measuring the Web(3/14)

- **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：

    Fraction of URLs sample from E2 found in E1

    --

    Fraction of URLs sample from E1 found in E2

Measuring the Web(4/14)

- Implementation
  - Building the Lexicon
  - Query based sampling
  - Query based checking
  - Bias

Measuring the Web(5/14)

- Building the Lexicon
  - For this experiment, a crawl of 300,000 documents in the Yahoo! hierarchy was used to build a lexicon of about 400,000 words
  - Low frequency words were NOT included
- 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

Measuring the Web(6/14)

- Query based checking
  - To test whether a search engine has indexed a given URL, we construct a query to check
  - Ideally, this query uniquely identifies the URL
  - But, there maybe be multiple results
    - multiple aliases or mirror copies
    - Normalization – all URLs are translated to lower case and all relative references and port numbers are removed
    - Actual Matching – this can be done multiple ways: Full URL, high similarity, weak URL, non-zero set

Measuring the Web(7/14)

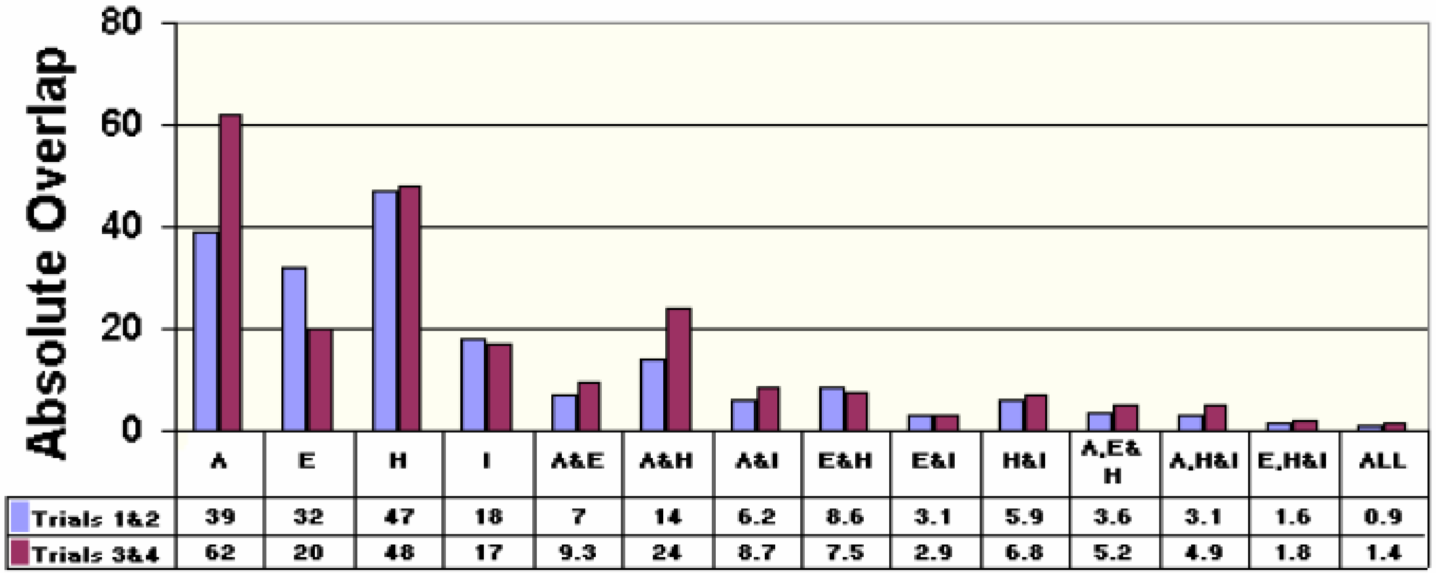
- Bias
  - Query Bias – favors large content rich documents
  - Ranking bias – introduced by search engines ranking pages. Only subsets are served up by the search while the remaining pages are not sampled.
  - Checking Bias – the method of matching and policy towards dynamic and low content pages influence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

Measuring the Web(8/14)

- Bias
  - Experimental bias – pages might be added and/or changed during the experiments, and search engines might under load or time-off queries
  - Malicious bias – some engines might choose not to serve pages that other pages have

Measuring the Web(9/14)

- 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(10/14)

- 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

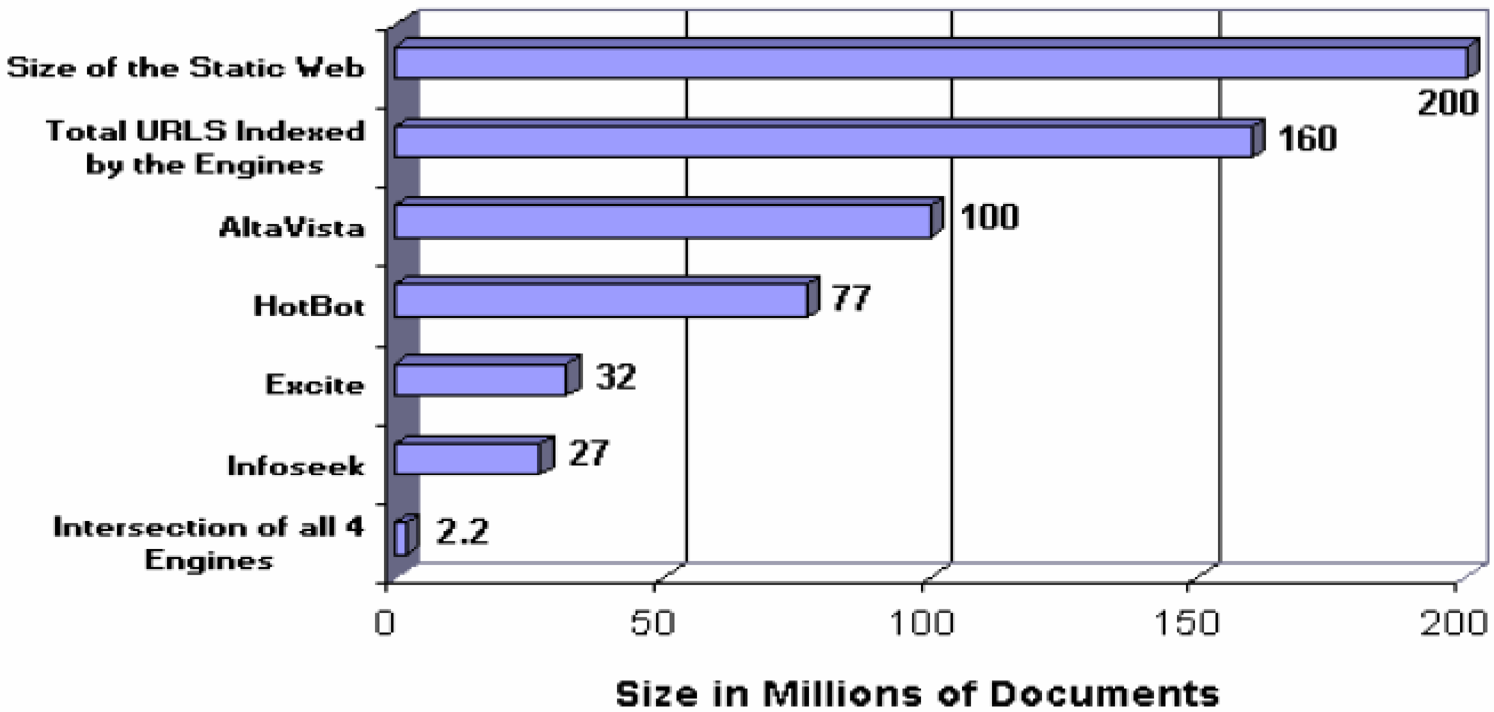


Figure 9: Absolute size estimates for November 1997.

Measuring the Web(11/14)

- Silverstein 98
  - 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.

Measuring the Web(12/14)

- Fully 15% of all request were empty.
- 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(13/14)

- 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(14/14)

- 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

Conclusion

- HITS algorithm and PageRanking algorithm are two most important algorithms in the search engines.
- Load balancing 、duplicate elimination
- Codir system
- Hierarchical directories
- Heuristic approach to measure the Web