

Optimized Query Execution in Large Search Engines with Global Page Ordering

Xiaohui Long Torsten Suel

CIS Department
Polytechnic University
Brooklyn, NY 11201

The Problem:

“how to optimize query throughput in large search engines, when the ranking function is a combination of term-based ranking and a global ordering such as Pagerank”

Talk Outline:

- **intro:** query processing in search engines
- **related work:** query execution and pruning techniques
- **algorithmic techniques**
- **experimental evaluation:** single and multiple nodes
- **concluding remarks**

Query processing in search engines (single node case)

- **inverted index**
 - a data structure for supporting text queries
 - like index in a book

disks with documents



indexing

aalborg	3452, 11437,
...	...
arm	4, 19, 29, 98, 143, ...
armada	145, 457, 789, ...
armadillo	678, 2134, 3970, ...
armani	90, 256, 372, 511, ...
...	...
zz	602, 1189, 3209, ...

- **Boolean queries:**

(zebra AND armadillo) OR alligator
 unions/intersections of lists

inverted index

Ranking in search engines:

- **scoring function:** assigns score to each document with respect to a given query
- **top-k queries:** return k documents with highest scores
- **example cosine measure for query with terms t_0 to t_{m-1}**

$$F(d, t_0, \dots, t_{m-1}) = \sum_{i=0}^{m-1} \frac{w(q, t_i) \cdot w(d, t_i)}{\sqrt{|d|}}$$

$$w(q, t) = \ln(1 + N/f_t), \text{ and}$$

$$w(d, t) = 1 + \ln f_{d,t}$$

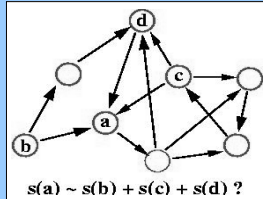
- can be implemented by computing score for all documents that contain any of the query words (union of inverted lists)
- in case of search engines: often intersection instead of union
- in large collections, lists are many MB for average queries

Pagerank Algorithm:

(Brin/Page 1998)

Basic Idea:

“significance of a page is determined by the significance of the pages linking to it”



More precisely:

- prune graph until no nodes with out-degree 0
- initialize every node with value $s(p) = 1/N$ (N number of nodes)
- iterate, where in each iteration we update each node as follows:

$$s(p) = (1 - \beta) \cdot \sum_{q \rightarrow p} \frac{s(q)}{d(q)} + \frac{\beta}{N}$$

where $d(q)$ is the out-degree of q and $\beta = 0.15$

- corresponds to random walk with random jump with prob. β

Combining Term-Based Methods and Pagerank

- recall the cosine measure:

$$F(d, t_0, \dots, t_{m-1}) = \sum_{i=0}^{m-1} \frac{w(q, t_i) \cdot w(d, t_i)}{\sqrt{|d|}}$$

$$w(q, t) = \ln(1 + N/f_t), \text{ and}$$

$$w(d, t) = 1 + \ln f_{d,t}$$

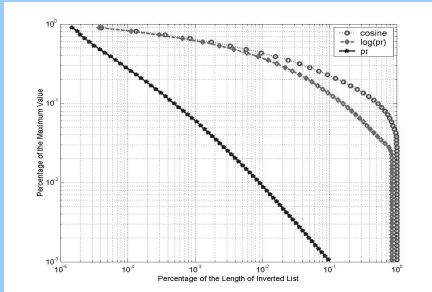
- Pagerank assigns a fixed score to each page (independent of query)
- naive way of integrating Pagerank value:

$$F(d, t_0, \dots, t_{m-1}) = g(d) + \sum_{i=0}^{m-1} f(d, t_i)$$

- works for any global ordering of page scores (e.g., based on traffic)
- but some more details remain

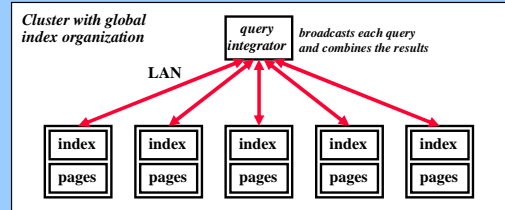
Using Pagerank in ranking:

- how to combine/add pagerank score and cosine? (addition)
- use PR or log(PR) ?
- normalize using mean of top-100 in list (Richardson/Domingo)



Query Processing in Parallel Search Engines

- low-cost cluster architecture (usually with additional replication)



- local index: every node stores and indexes subset of pages
- every query broadcast to all nodes by query integrator (QI)
- every node supplies top-10, and QI computes global top-10
- note: we don't really need top-10 from all, maybe only top-2

Related Work on top-k Queries

- IR: optimized evaluation of cosine measures (since 1980s)
- DB: top-k queries for multimedia databases (Fagin 1996)
- does not consider combinations of term-based and global scores
- Brin/Page 1998: fancy lists in Google

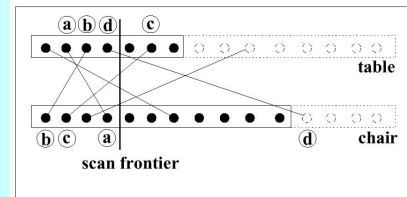
Related Work (IR)

- basic idea: "presort entries in each inverted list by contribution to cosine"
- also process inverted lists from shortest to longest list
- various schemes, either reliable or probabilistic
- most closely related:
 - Persin/Zobel/Sacks-Davis 1993/96
 - Anh/Moffat 1998, Anh/deKretzer/Moffat 2001
- typical assumptions: many keywords/query, OR semantics

Related Work (DB)

(Fagin 1996 and others)

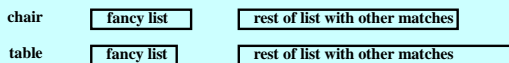
- motivation: searching multimedia objects by several criteria
- typical assumptions: few attributes, OR semantics, random access
- FA (Fagin's algorithm), TA (Threshold algorithm), others
- formal bounds: $N^{m-1/k} \cdot k^{1/m}$ for k lists if lists independent
- term-based ranking: presort each list by contribution to cosine



Related Work (Google)

(Brin/Page 1998)

- "fancy lists" optimization in Google
- create extra shorter inverted list for "fancy matches" (matches that occur in URL, anchor text, title, bold face, etc.)



- note: fancy matches can be modeled by higher weights in the term-based vector space model
- no details given or numbers published

Results of our Paper

- pruning techniques for query execution in large search engines
- focus on a combination of a term-based and a global score (such as Pagerank)
- techniques combine previous approaches such as fancy lists and presorting of lists by term scores
- experimental evaluation on 120 million pages
- very significant savings with almost no impact on results
- it's good to have a global ordering!

Algorithms:

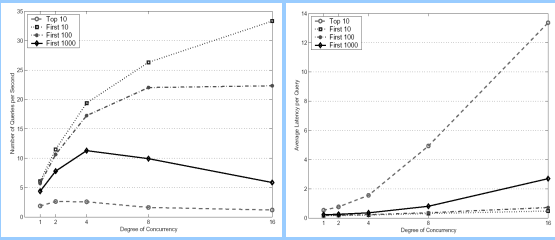
- *exhaustive algorithm*: “no pruning, traverse entire list”
- *first-m*: “a naïve algorithm with lists sorted by Pagerank; stop after m elements in intersection found”
- *fancy first-m*: “use fancy and non-fancy lists, each sorted by Pagerank, and stop after m elements found”
- *reliable pruning*: “stop when top-k results found”
- *fancy last-m*: “stop when at most m elements unresolved”
- single-node and parallel case with optimization

Experimental setup:

- 120 million pages on 16 machines (1.8TB uncompressed)
- P-4 1.7Ghz with 2x80GB Seagate Barracuda IDE
- compressed index based on Berkeley DB (using the *mg* compression macros)
- queries from Excite query trace from December 1999
- queries with 2 terms in the following
- local index organization with query integrator
- first results for one node (7.5 million pages), then 16
- note: do not need top-10 from every node
- motivates top-1, top-4 schemes and precision at 1, 4
- ranking by *cosine + log(PR)* with normalization

A naïve approach: *first-m*

- sort inverted lists by Pagerank (docID = rank due to Pagerank)
- *exhaustive*: top-10
- *first-m*: return 10 highest scoring among first 10/100/1000 pages in intersection

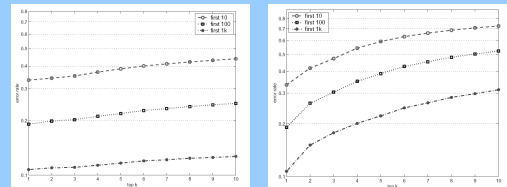


first-m (ctd.)

top-10	first-1000	first-100	first-10
417.6	110.8	41.0	14.0

average cost per query in terms of disk blocks

loose/strict precision, relative to “correct” cosine + log(PR)



- for first-10, about 45% of top-10 results belong in top-10
- for first-100, about 80% of queries return correct top-1 result
- for first-1000, about 85% of top-10 results belong in top-10
- for first-1000, about 70% of queries return all correct top-10 results

How can we do better?

(1) Use better stopping criteria?

- reliable pruning: stop when we are sure
- probabilistic pruning: stop when almost sure
- do not work well for Pagerank-sorted index

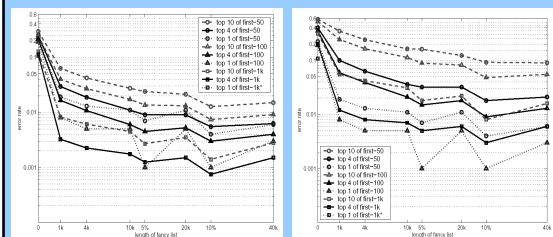
(2) Reorganize index structure?

- sort lists by term score (cosine) instead of Pagerank
- does not do any better than sorting by Pagerank only
- sort lists by term + 0.5 log(PR) (or some combination of these)
- some problems in normalization and dependence on # of keywords
- *generalized fancy lists*
- for each list, put entries with highest term value in fancy list
- sort both lists by pagerank docID
- note: anything that does well in 2 out of 3 scores is found soon
- deterministic or probabilistic pruning, or first-k

chair	fancy list	rest of list, cosine < x
table	fancy list	rest of list, cosine < y

Results for generalized fancy lists

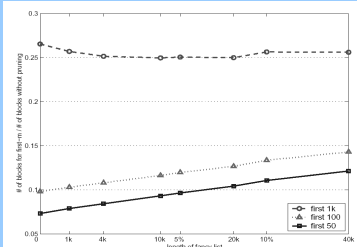
- loose vs. strict precision for various sizes of the fancy lists



- MUCH better precision than without fancy lists!
- for first-1000, we always get correct top-1 in these runs

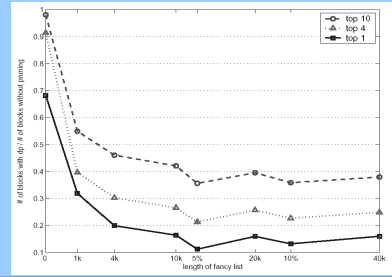
Costs of Fancy Lists

- cost similar to *first-m* without fancy lists plus the additional cost of reading fancy lists
- cost increases slightly with size of fancy list
- slight inefficiency: fancy list items not removed from other list
- note: we do not consider savings due to caching

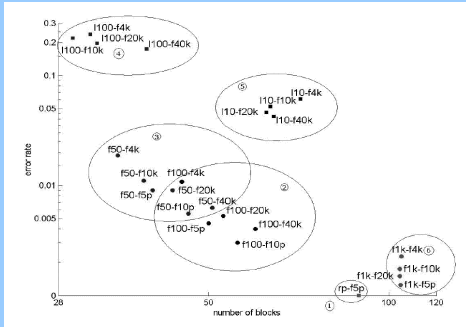


Reliable Pruning

- always gives “correct” result
- top-4 can be computed reliably with ~20% of original cost
- with 16 nodes, top-4 from each node suffice with 99% prob. to get top-10



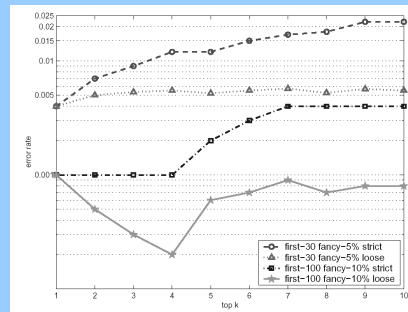
Summary for all Approaches (single node)



comparison of first-m, last-m, and reliable pruning schemes for top-4 queries with loose precision

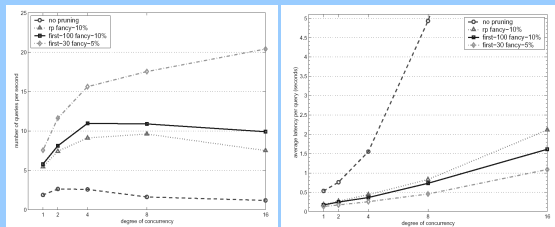
Results for 16 Nodes

- first-30 returns correct top-10 for almost 98% of all queries



Throughput and Latency for 16 Nodes

- top-10 queries on 16 machines with 120 million pages
- up to 10 queries/sec with reliable pruning
- up to 20 queries per second with first-30 scheme



Note: reliable pruning not implemented in purely incremental manner

Current and Future Work

- results for 3+ terms and incremental query integrator
- need to do precision/recall study
- need to engineer ranking function and reevaluate
- how to include term distance in documents
- impact of caching at lower level
- working on publicly available engine prototype
- tons of loose ends and open questions