5 Index Pruning

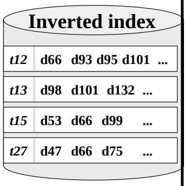
5.1 Index-based Query Processing
5.2 Pruning with Combined Authority/Similarity Scoring
5.3 Pruning with Similarity Scoring

5.1 Index-based Query Processing

Given: query q = t1 t2 ... tz with z (conjunctive) keywords
document collection D = {d1, ..., dn} of m-dim. vectors dj
similarity scoring function score(q,d)
Find: top k results with regard to score(q,d)

Inverted index contains for each ti an ***index list***

- with all docs that contain ti, sorted by doc id
- each ti has idf, each doc id has tf in index list
- index is organized as a search tree (usually B+-tree or suffix tree/trie)
- index lists are only scanned, no random access



Inverted index

t12	**d66 d93 d95 d101 ...**
t13	**d98 d101 d132 ...**
t15	**d53 d66 d99 ...**
t27	**d47 d66 d75 ...**

Naive QP algorithm:

```
candidate-docs := ∅;
for i=1 to z do {
    candidate-docs := candidate-docs ∪ index-lookup(ti) };
for each dj ∈ candidate-docs do {compute score(q,dj)};
sort candidate-docs by score(q,dj) descending;
```

Framework for Index Pruning

Goal: avoid scanning very long index lists until completion

Assume scoring of the form $$\boldsymbol{score(q,d_j) = \sum_{i=1}^{z} s_i(t_i,d_j) \;+\; r(d_j)}$$

Key ideas:

1) keep index lists in specific sort order
2) possibly keep redundant lists (possibly in alternative sort order)
3) accumulate partial scores for docs by adding up s_i or r values
4) while scanning the z query-relevant index lists maintain:
   pos(i) – current position in the i-th index list
   high(i) – upper bound for all docs in i-th list that follow pos(i)
   high(0) – upper bound for r(dj) among all docs not in current top k
   high – upper bound for all docs with incomplete score
5) in each step compute high(i) and high,
   and compare to min score of current top k
   stop scanning i-th index list if high(i) < min score of top k

5.2 Pruning with Combined Authority/Similarity Scoring (Long/Suel 2003)

Focus on score(q,dj) = r(dj) + s(q,dj)
with normalization r(·) ≤ a, s(·) ≤ b (and often a+b=1)
Keep index lists sorted in descending order of authority r(dj)

Authority-based pruning (used in Google ?):

```
high(0) := max{r(pos(i))|i=1..z}; high := high(0)+b;
high(i) := r(pos(i))+b;
stop scanning i-th index when high(i) < min score of top k
```

effective when total score of top k results is dominated by r

First-m heuristics (used in Google ?):

```
scan all z index lists until m ≥ k docs have been found
that appear in all lists;
```

the stopping condition is easy to check because of the sorting by r

Separating Documents with Large s_i Values

Idea (Google):
in addition to the full index lists L(i) sorted by r,
keep short ***„fancy lists“*** F(i) that contain the docs dj
with the highest values of si(ti,dj) and sort these by r

Fancy first-m heuristics:

```
Compute total score for all docs in ∩ F(i) (i=1..k)
    and keep top k results;
Cand := ∪_i F(i) − ∩_i F(i);
for each dj ∈ Cand do {compute partial score of dj};
Scan full index lists L(i) (i=1..k);
    if pos(i) ∈ Cand
        {add $s_i$(ti,pos(i)) to partial score of pos(i)}
    else {add pos(i) to Cand and set its partial score to $s_i$(ti,pos(i))};
    Terminate the scan when m docs
    have a completely computed total score;
```

Authority-based Pruning with Fancy Lists

Guarantee that the top k results are complete by
extending the fancy first-m heuristics as follows:
stop scanning the i-th index list L(i) not after m results,
but only when we know that no imcompletely scored doc
can qualify itself for the top k results

Maintain:

```
r_high(i) := r(pos(i))
s_high(i) := max{si(q,dj) | dj ∈ L(i) – F(i)}
```

Scan index lists L(i) and accumulate partial scores for all docs dj
Stop scanning L(i) iff
$r\_high(i) + \Sigma_i\, s\_high(i) < \min\{score(d) \mid d \in \text{current top k results}\}$

Probabilistic Pruning

Maintain statistics about the distribution of si values

For pos(i)

estimate the probability p(i) that the rest of L(i) contains a doc d for which the si score is so high that d qualifies for the top k results

Stop scanning L(i) if p(i) drops below some threshold

Simple „approximation“ by the ***last-l heuristics***:

stop scanning when the number of docs in $\cup_i$ F(i) $-\cap_i$ F(i) with incompletely computed score drops below l (e.g., l=10 or 100)

Performance Experiments (1)

Setup:

index lists for 120 Mio. Web pages distributed over 16 PCs
(and stored in BerkeleyDB databases)

query evaluation iterated over many sample queries
with different degrees of concurrency (multiprogramming levels)

Evaluation measures:

query throughput [queries/second]

average query response time [seconds]

error for pruning heuristics:

strict-k error: fraction of queries for which the top k were not exact

loose-k error: fraction of top k results that do not belong to true top k

Performance Experiments (2)

Exact computation versus first-m heuristics (without fancy lists)

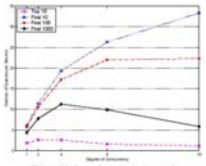


Figure 6.1: Query throughput per second.

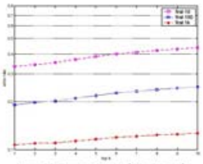


Figure 6.3: Error rate for loose-*k*, for different *k*.

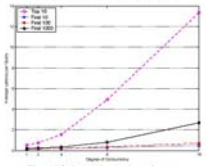


Figure 6.2: Average latency for each query.

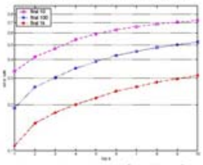
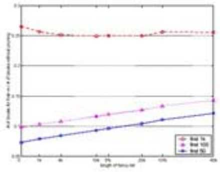


Figure 6.4: Error rate for strict-*k*, for different *k*.

from: X. Long, T. Suel, Optimized Query Execution in Large Search Engines with Global Page Ordering, VLDB 2003

Performance Experiments (3)

Fancy first-m heuristics



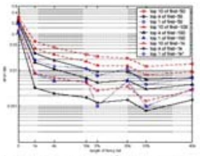


Figure 6.6: Error rate for loose-*k* measure, for different values of *m* and *k*, and different lengths of the fancy lists.

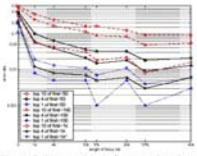


Figure 6.7: Error rate for strict-*k* measure, for different values of *m* and *k*, and different lengths of the fancy lists.

from: X. Long, T. Suel, Optimized Query Execution in Large Search Engines with Global Page Ordering, VLDB 2003

Performance Experiments (4)

Authority-based pruning with fancy lists

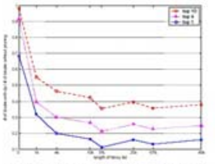


Figure 6.8: Ratio of the number of blocks accessed by reliable pruning and the baseline scheme, for different *k* and different lengths of the fancy lists.

Overall comparison of pruning techniques

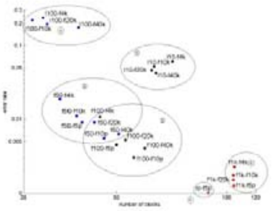


Figure 6.9: Comparison of various pruning techniques.

from: X. Long, T. Suel, Optimized Query Execution in Large Search Engines with Global Page Ordering, VLDB 2003

Performance Experiments (5)

Scalability on 16-node cluster
(with distributed index lists and centralized QP on one node)

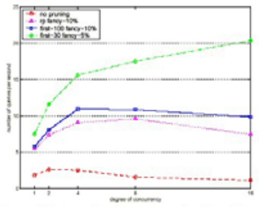


Figure 6.10: Query throughput per second on **16** machines.

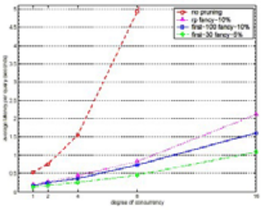


Figure 6.11: Query latency in seconds on **16** machines.

from: X. Long, T. Suel, Optimized Query Execution in Large Search Engines with Global Page Ordering, VLDB 2003

5.3 Pruning with Similarity Scoring

Focus on scoring of the form $\boldsymbol{score(q,d_j) = \sum_{i=1}^{z} s_i(t_i,d_j)}$

with $\boldsymbol{s_i(t_i,d_j) = tf(t_i,d_j) \cdot idf(t_i) \cdot idl(d_j)}$

Implementation based on a hash array of *accumulators*
for summing up the partial scores of candidate results

quit heuristics
(with doc-id-ordered or tf-ordered or tf*idl-ordered index lists):
ignore index list L(i) if idf(ti) is below threshold
or stop scanning L(i) if idf(ti)*tf(ti,dj)*idl(di) drops below threshold
or stop scanning L(i) when the number of accumulators is too high

continue heuristics:
upon reaching threshold, continue scanning index lists,
but do not add any new documents to the accumulator array

Greedy QP

Assume index lists are sorted by tf(ti,dj) (or tf(ti,dj)*idl(dj)) values

```
Open scan cursors on all z index lists L(i)
Repeat
    Find pos(g) among current cursor positions pos(i) (i=1..z)
        with the largest value of idf(ti)*tf(ti,dj)
        (or idf(ti)*tf(ti,dj)*idl(dj));
    Update the accumulator of the corresponding doc;
    Increment pos(g);
Until stopping condition
```

QP with Compressed Index Lists (Moffat/Zobel 1996)

Keep L(i) in ascending order of doc id‘s
Compress L(i) by actually storing the gaps between successive doc id‘s
(or using some more sophisticated prefix-free code)

QP may start with those L(i) lists which are short and have high idf
Candidate results need to be looked up in other lists L(j)
To avoid having to uncompress the entire list L(j),
L(j) is encoded into groups of entries
with a skip pointer at the start of each group

Literature

- Xiaohui Long, Torsten Suel: Optimized Query Execution in Large Search Engines with Global Page Ordering, VLDB Conf., 2003
- Alistair Moffat, Justin Zobel: Self-Indexing Inverted Files for Fast Text Retrieval, ACM TOIS Vol.14 No.4, 1996
- Vo Ngoc Anh, Owen de Kretser, Alistair Moffat: Vector-Space Ranking with Effective Early Termination, SIGIR Conf., 2001
- Vo Ngoc Anh, Alistair Moffat: Impact Transformation: Effective and Efficient Web Retrieval, SIGIR Conf., 2002
- Norbert Fuhr, Norbert Gövert, Mohammad Abolhassani: Retrieval Quality vs. Effectiveness of Relevance-oriented Search in XML Documents, Technical Report, University of Duisburg, 2003