Efficient Top-K Query Processing
for Text, Semistructured-, and Structured Data

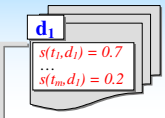
Martin Theobald
mtb@mpi-inf.mpg.de

1

Threshold Algorithms for Top-k Retrieval

R.Fagin et al., PODS '01
Balke et al., VLDB '00

Corpus: $d_1,\ldots,d_n$



Query: $q = (t_1,t_2,t_3)$

INDEXING

Inverted Index

L_1	d78 0.9	d23 0.8	d10 0.8	d1 0.7	d88 0.2 …
L_2	d64 0.8	d23 0.6	d10 0.6	d10 0.2	d78 0.1 …
L_3	d10 0.7	d78 0.5	d64 0.4	d99 0.2	d34 0.1 …

1. ***NRA(q,L):***
2. scan all lists L_i (i = 1..m) in parallel & consider doc d at pos_i
3. E(d) := E(d) ∪ {i};
4. $high_i = s_i(d)$;
5. **worstscore(d)** := $\sum\{s_v(d) \mid v \in E(d)\}$;
6. **bestscore(d)**:= worstscore(d) + $\sum\{high_v \mid v \notin E(d)\}$;
7. if worstscore(d) > min-k then
8. add d to top-k
9. **min-k** := min{worstscore(d') | d' ∈ top-k};
10. else if bestscore(d) > min-k thens
11. candidates := candidates ∪ {d};
12. if **max {bestscore(d') | d' ∈ candidates} ≤ min-k** then exit;

k = 1

Scan depth 3

Rank	*Doc*	*Worst-score*	*Best-score*
1	d10	2.1	2.1
2	d78	1.4	2.0
3	d23	1.4	1.8
4	d64	1.2	2.0

STOP!

2

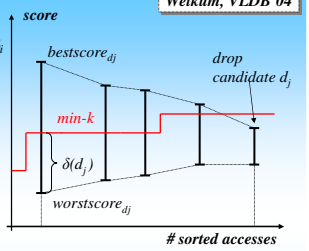
Prob-k – Probabilistic Candidate Pruning

Theobald, Schenkel, Weikum, VLDB'04

TA family of algorithms based on invariant:

$$\sum_{i \in E(d_j)} s_i(d_j) \le score(q,d_j) \le \sum_{i \in E(d_j)} s_i(d_j) + \sum_{i \notin E(d_j)} high_i$$

- If worstscore(d_j) > min-k → add d_j to top-k
- If bestscore(d_j) ≤ min-k → drop d_j from queue



Probabilistic threshold test:

If $p(d_j) = P\left[\sum\{s_i \mid i \notin E(d_j)\} > \delta(d_j)\right] < \varepsilon$

with $\delta(d_j) := \min_k - \sum\{s_i \mid i \in E(d_j)\}$

→ Drop d_j from queue

Score predictors for different distributions:

- ***Uniform*** in *[0, $high_i$]* or in *[low_i, $high_i$]*
- ***Truncated Poisson*** over n_i equidistant values with $P[d = v_j] = e^{-\alpha}\frac{\alpha^{j-1}}{(j-1)!}$
- ***Arbitrarily distributed*** and approximated by an ***equi-width histogram*** with cells j=1,..,c ranging over $(l_j, h_j]$

→ **Consider convolution of score distributions over all f_i for all $i \notin E(d_j)$**

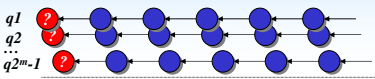
3

Prob-k – Efficient Queue Management

Theobald, Schenkel, Weikum, VLDB'04

Prob-conservative

- 2^m **-1 queues** per query & use bestscore(d) as priority
- Group candidates by remainder sets *[1..m] – E(d)*
- Top candidate dominates all candidates in each queue
- Test top candidate only
- For all queues *q* : **Drop queue *q***, if
  *P[**top(q)** can qualify for top-k] ≤ ε*



Prob-progressive

- ***1 queue*** per query & use bestscore(d) as priority
- Merge all candidates by their best-scores
- No dominating candidate wrt. score prediction
- Test all candidates periodically
- For all candidates *d* in *q* :
  Drop **individual candidate *d***, if
  *P[**d** can qualify for top-k] ≤ ε*

Prob-smart

- ***1 bounded queue*** per query & use bestscore(d) as priority
- Merge all candidates by their best-scores
- No dominating candidate in terms of score prediction
- Update & rebuild entire queue periodically
- **Stop heuristically**, if
  *P[**top(q)** can qualify for top-k] ≤ ε*

Prob-aggressive

- ***No queue***
- Use efficient hash-joins only
- Consider virtual candidate d_v with E(d_v) = ∅
- d_v dominates all yet unseen candidates
- **Stop heuristically**, if
  P[d_v can qualify for top-k] ≤ ε

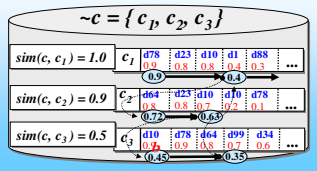
4

Self-tuning Incremental Query Expansions

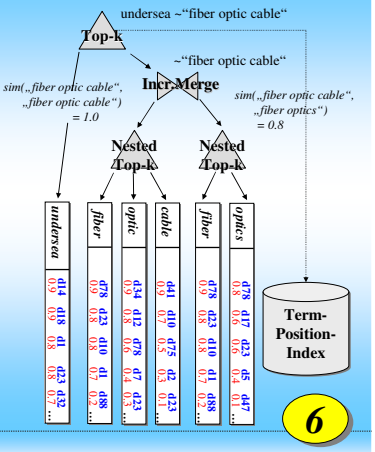
Theobald, Schenkel, Weikum, SigIR'05

Incremental Merge operator

- **Incrementally merge** $L_{i1},..L_{ik}$ in descending order of s_{ij} into a single virtual index list
- Seamless integration into top-k operator and single threshold termination
- Sharp semantics:
  *"**The best match** out of $c_1,..c_k$"* instead of
  "Any or all of $c_1,..c_k$"
- Modified score aggregation
  $score(d_j) := \sum_{i=1}^{m} \max_{c_{ip} \in \exp(c_i)} \{s(c_{ip},d_j)\}$
- Incorporate additional similarity measures
  → ***sim(c,c_i)*s_{ij}** (e.g., ontological similarities)
- **Increased efficiency and retrieval robustness!**



Nested Top-k operator

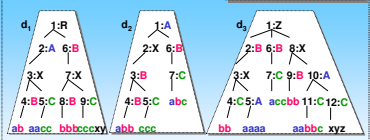


5

TopX – Efficient Support for XML IR

Theobald, Schenkel, Weikum, VLDB'05

- *Supports the* XPath 2.0 Full-Text *and* NEXI *query languages*
  //article[about(.//(abs|kwd), "genetic algorithm")]
  //bdy//sec[about(simulated annealing -fuzzy)]
- *Probabilistic candidate pruning*
  - Combined aggregated score- and XML-specific selectivity predictor
- *XML-specific scoring model*
  - Okapi BM-25 extension for semistructured data using individual element statistics
  - Optional redundant full-content indexing
  - Accelerates **descendants** axis between tag/term pairs and/or tag paths
- *Supports multiple index structures*
  - Pre/Post-Order or Data Guides
  - Dynamic query rewriting wrt. to individual predicate selectivities
- *Cost-based scheduling of random I/Os to test navigational query conditions*



Pre-computed index table:

Tag	Term	MaxScore	DocId	Score	ElemId	Pre	Post
A	a	1	d3	1	e5	5	2
A	a	1	d3	1/4	e10	10	9
A	a	1/2	d1	1/2	e2	2	4
A	a	2/9	d2	2/9	e1	1	3
B	b	1	d1	1	e6	6	5
B	b	1	d1	1/2	e4	4	7
B	b	1	d3	3/7	e6	6	8
B	b	1	d3	1	e9	9	7
B	b	1/3	d3	1/3	e2	2	4
B	b	2/3	d2	2/3	e4	4	1
B	b	2/3	d2	1/3	e3	3	3
B	b	2/3	d2	1/3	e6	6	6
C	c	1	d2	1	e5	5	2
C	c	1	d2	1/3	e7	7	5
C	c	2/3	d3	2/3	e7	7	5
C	c	2/3	d3	1/5	e11	11	8
C	c	3/5	d1	3/5	e9	9	6
C	c	3/5	d1	1/2	e5	5	2

B⁺-tree supported block-scans:

~~(A, a, d3, ...)~~
(B, b, d1, ...)
(C, c, d2, ...)
(A, a, d1, ...)
~~(B, b, d3, ...)~~
(C, c, d3, ...)
(A, a, d2, ...)
(B, b, d2, ...)
(C, c, d1, ...)

Sorted Block-Scans & Random Range-Scans

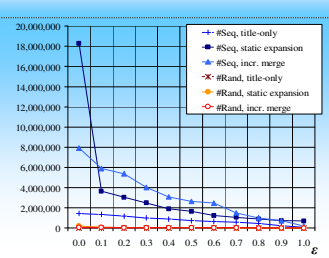
Experiments

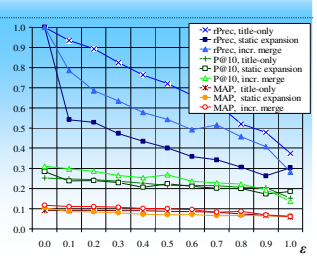
– **Aquaint News Corpus:** 528,155 docs; ~86,000,000 tuples; 1.9 GB
– 50 "hard" queries taken from the **TREC Robust track 2004**

Prob. Pruning Performance

Incremental Merge vs. Static Expansions

m ≤ 118
k = 10 for P@10
k = 1,000 for MAP

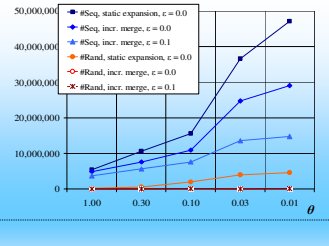


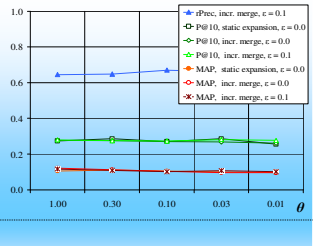


Expansion Performance

Incremental Merge vs. Static Expansions

36 ≤ m ≤ 876
k = 10 for P@10
k = 1,000 for MAP





MAX-PLANCK-GESELLSCHAFT

max planck institut informatik