Chapter 12: Query Processing

Computers are useless,
they can only give you answers.

-- Pablo Picasso

You have to think anyway,
so why not think big?

-- Donald Trump

There are lies, damn lies,
and workload assumptions.

-- anonymous

Outline

12.1 Query Processing Algorithms

12.2 Fast Top-k Search

12.3 Phrase and Proximity Queries

12.4 Query Result Diversification

loosely following Büttcher/Clarke/Cormack Chapters 5 and 8.6
plus Manning/Raghavan/Schütze Chapters 7 and 9
plus specific literature

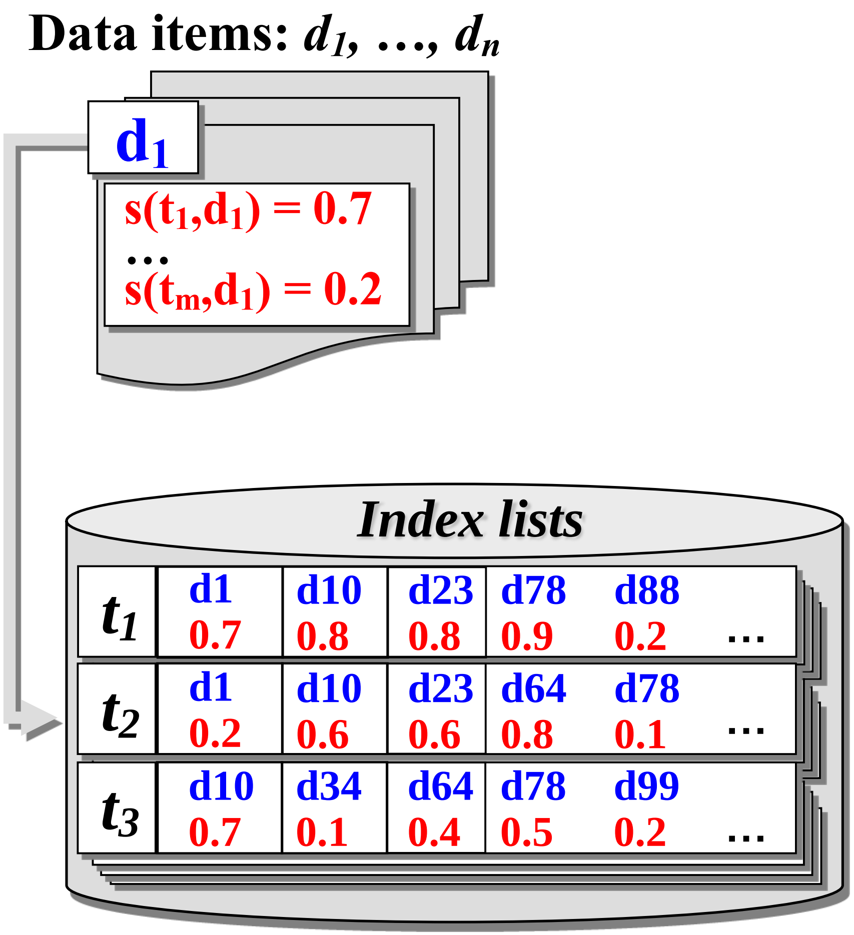
Query Types

- **Conjunctive**
  (i.e., all query terms are required)
- **Disjunctive**
  (i.e., subset of query terms sufficient)
- **Phrase** or **proximity**
  (i.e., query terms must occur in right order or close enough)
- **Mixed-mode** with **negation**
  (e.g., *"harry potter" review +movie -book*)
- Combined with **ranking of result documents** according to

  $$score(q,d) = \sum_{t \in q} score(t,d)$$

  with *score*(t, d) depending on retrieval model (e.g. tf*idf)

Indexing with Document-Ordered Lists



index-list entries stored in ascending order of document identifiers (**document-ordered lists**)

process all queries (conjunctive/disjunctive/mixed) by sequential scan and merge of posting lists

Document-at-a-Time Query Processing

Document-at-a-Time (DAAT) query processing

- assumes **document-ordered posting lists**
- scans posting lists for query terms $t_1, \ldots, t_{|q|}$ **concurrently**
- maintains an **accumulator** for each candidate result doc:
- $acc(d) = \sum_{i:\, d \text{ seen in } L(ti)} score(ti, d)$

a	d_1, 1.0	d_4, 2.0	d_7, 0.2	d_8, 0.1
b	d_4, 1.0	d_7, 2.0	d_8, 0.2	d_9, 0.1
c	d_4, 3.0	d_7, 1.0		

Accumulators

d_1	:	1.0
d_4	:	6.0
d_7	:	3.2
d_8	:	0.3
d_9	:	0.1

- always advances posting list with **lowest current doc id**
- exploit **skip pointers** when applicable
- required **memory** depends on **# results** to be returned
- **top-k results** in **priority queue**

DAAT with Weak And: WAND Method

[Broder et al. 2003]

Disjunctive (Weak And) query processing

- assumes document-ordered posting lists with known **maxscore(i) values** for each t_i: $\max_d$ (score (d,t_i))
- While scanning posting lists keep track of
  - **min-k:** the lowest total score in current top-k results
  - **ordered term list**: terms sorted by docId at current scan pos
  - **pivot term:** smallest j such that *min-k* $\leq \sum_{i \leq j} maxscore(i)$
  - **pivot doc:** doc id at current scan pos in posting list Lj

Eliminate docs that cannot become top-k results (**maxscore pruning**)

- **if** pivot term does not exist (min-k > $\sum_i maxscore(i)$)
- **then** stop
- **else** advance scan positions to pos ≥ id of pivot doc (**“big skip“**)

Example: DAAT with WAND Method

[Broder et al. 2003]

Key invariant: For terms i=1..|q| and current scan positions cur_i

assume that cur_1 = min {curi | i=1..|q|}

Then for each posting list i there is no docid between cur_1 and cur_i

$maxscore_i$	term i	cur_i				
5	1	…	101			
4	2	…	…	250		
2	3	…	…	…	300	
3	4	cannot contain any docid $\in$[102,599]				600

Suppose that min-k = 12

then the pivot term is 4

($\Sigma_{i=1.3}$ $maxscore_i$ > min-k, $\Sigma_{i=1.4}$ $maxscore_i$ $\le$ min-k)

and the pivot docid is 600

$\rightarrow$ can advance all scan positions cur_i to 600

Term-at-a-Time Query Processing

Term-at-a-Time (TAAT) query processing

- assumes **document-ordered posting lists**
- scans posting lists for query terms $t_1, \ldots, t_{|q|}$ **one at a time,** (possibly in decreasing order of idf values)
- maintains an **accumulator** for each candidate result doc
- after processing L(tj): $acc(d) = \sum_{i \leq j} score(ti, d)$

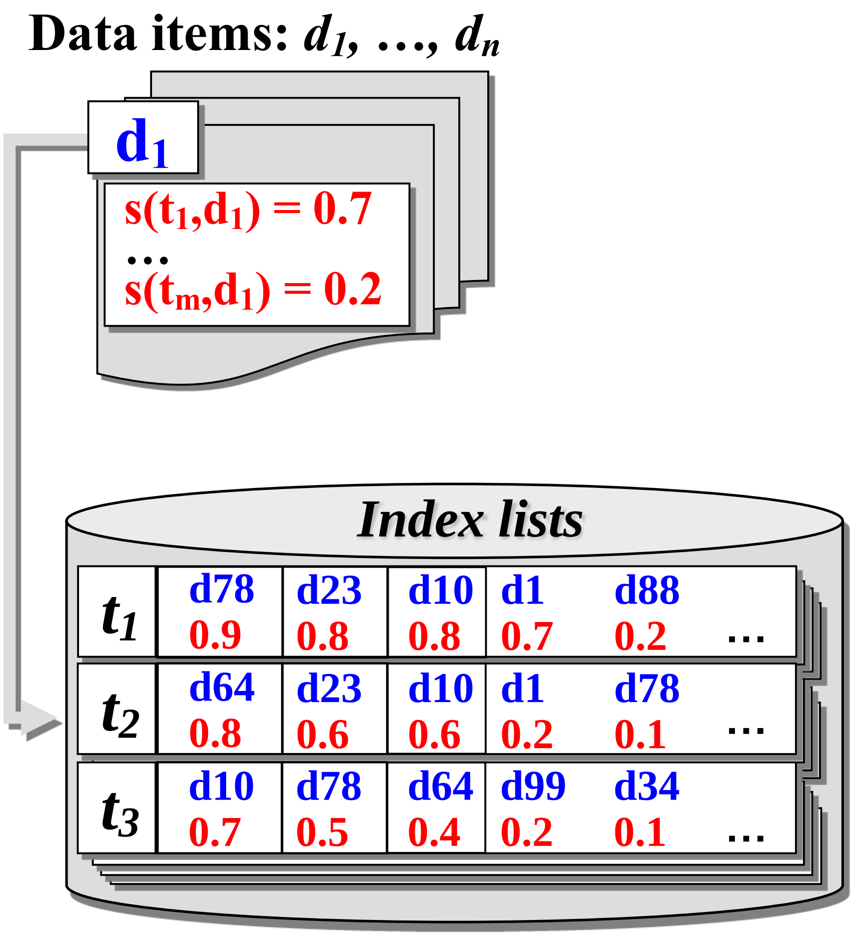
a ········	***d***d_1, 1.0	d_4, 2.0	d_7, 0.2	d_8, 0.1
b ········	d_4, 1.0	d_7, 2.0	d_8, 0.2	d_9, 0.1
c ········	d_4, 3.0	d_7, 1.0		

Accumulators

d_1	:	[illegible].0
d_4	:	[illegible].0
d_7	:	[illegible].[illegible]
d_8	:	0.[illegible]
d_9	:	0.[illegible]

- **memory** depends on the **number of accumulators** maintained
- TAAT is attractive when scanning many short lists

Indexing with Impact-Ordered Lists



index-list entries stored in descending order of per-term score impact (**impact-ordered lists**)

aims to avoid having to read entire lists
rather scan only (short) prefixes of lists

Greedy Query Processing Framework

Assume index lists are sorted by $tf(t_i,d_j)$ or $tf(t_i,d_j)*idl(d_j)$ values
idf values are stored separately

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

Stopping Criterion: Quit & Continue Heuristics

[Zobel/Moffat 1996]

For scoring of the form $$\text{score}(q, d_j) = \sum_{i=1}^{m} s_i(t_i, d_j)$$

with $$s_i(t_i, d_j) \sim tf(t_i, d_j) \cdot idf(t_i) \cdot idl(d_j)$$

Assume **hash array of *accumulators*** *for* summing up score mass of candidate results

***quit* heuristics** (with docId-ordered or tf-ordered or tf*idl-ordered index lists):

- ignore index list L(i) if idf(t_i) is below tunable threshold or
- stop scanning L(i) if idf(t_i)*tf(t_i,d_j)*idl(d_j) 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

12.2 Fast Top-k Search

Top-k aggregation query over relation ***R (Item, A1, ..., Am)***:

Select Item, s(R1.A1, ..., Rm.Am) As Aggr

From Outer Join R1, …, Rm Order By Aggr Limit k

with **monotone s**: $(\forall i: \ x_i \geq x_i') \Rightarrow s(x_1 \dots x_m) \geq s(x_1' \dots x_m')$

(example: item is doc, attributes are terms, attr values are scores)

- Precompute per-attr (index) **lists sorted in desc attr-value order** (score-ordered, impact-ordered)
- Scan lists by **sorted access (SA)** in round-robin manner
- Perform **random accesses (RA) by Item** when convenient
- Compute aggregation s **incrementally** in **accumulators**
- Stop when **threshold test** guarantees correct top-k (or when heuristics indicate „good enough“ approximation)

simple & elegant, adaptable & extensible to distributed system

following R. Fagin: Optimal aggregation algorithms for middleware, JCSS. 66(4), 2003

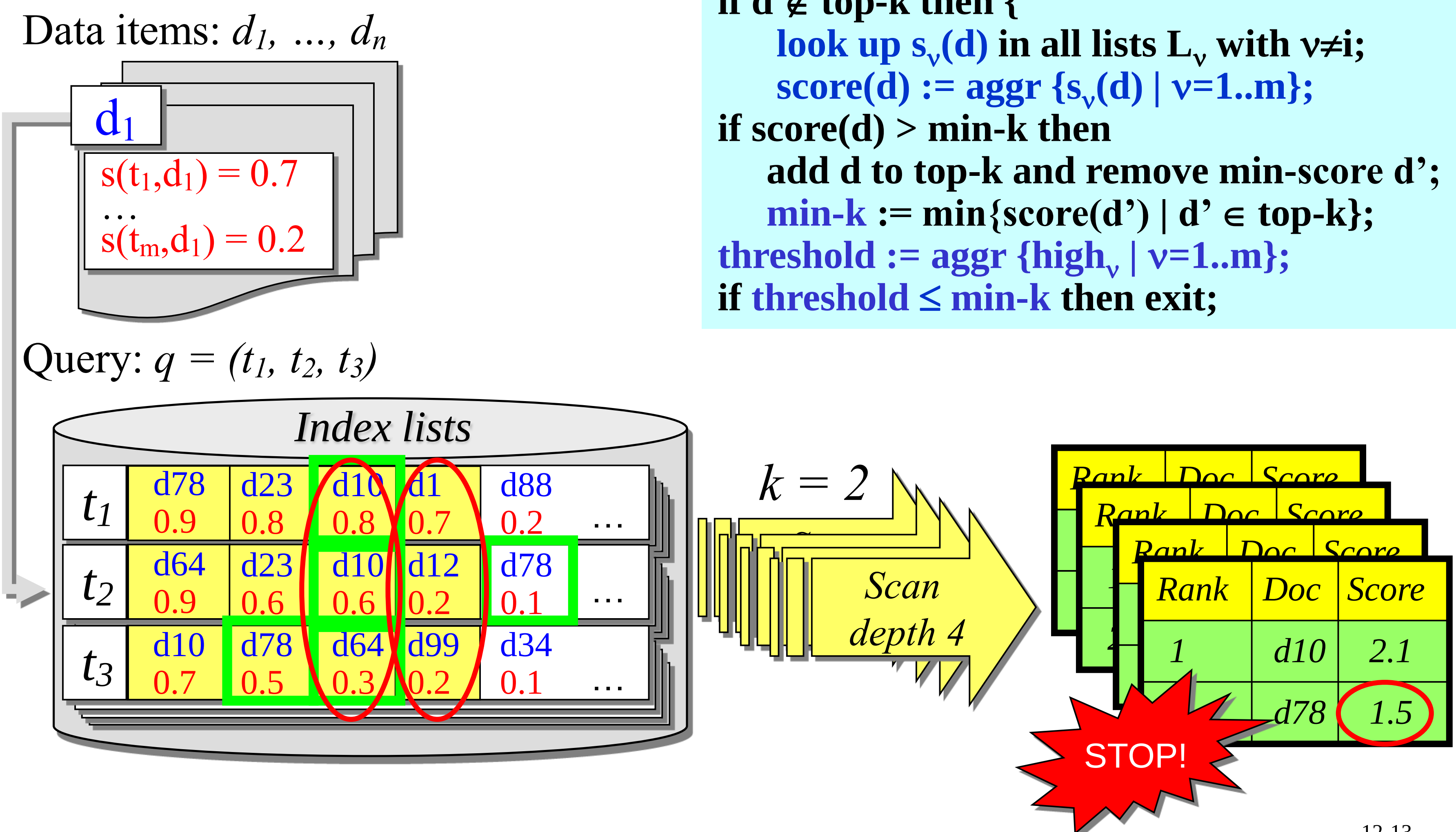
Threshold Algorithm (TA)

[Fagin 01, Güntzer 00, Nepal 99, Buckley 85]

simple & DB-style;
needs only O(k) memory

Threshold algorithm (TA):

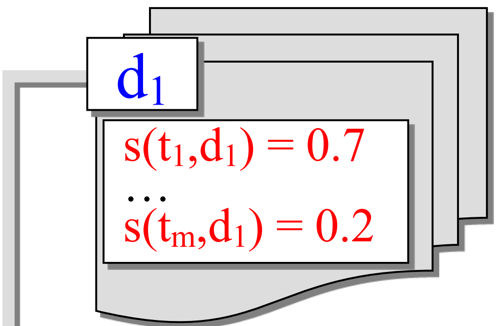
scan index lists; consider d at pos_i in L_i;
$high_i := s(t_i,d)$;
if d $\notin$ top-k then {
look up $s_\nu(d)$ in all lists L_ν with $\nu \neq i$;
score(d) := aggr $\{s_\nu(d) \mid \nu=1..m\}$;
if score(d) > min-k then
add d to top-k and remove min-score d';
min-k := min{score(d') | d' $\in$ top-k};
threshold := aggr $\{high_\nu \mid \nu=1..m\}$;
if threshold $\leq$ min-k then exit;



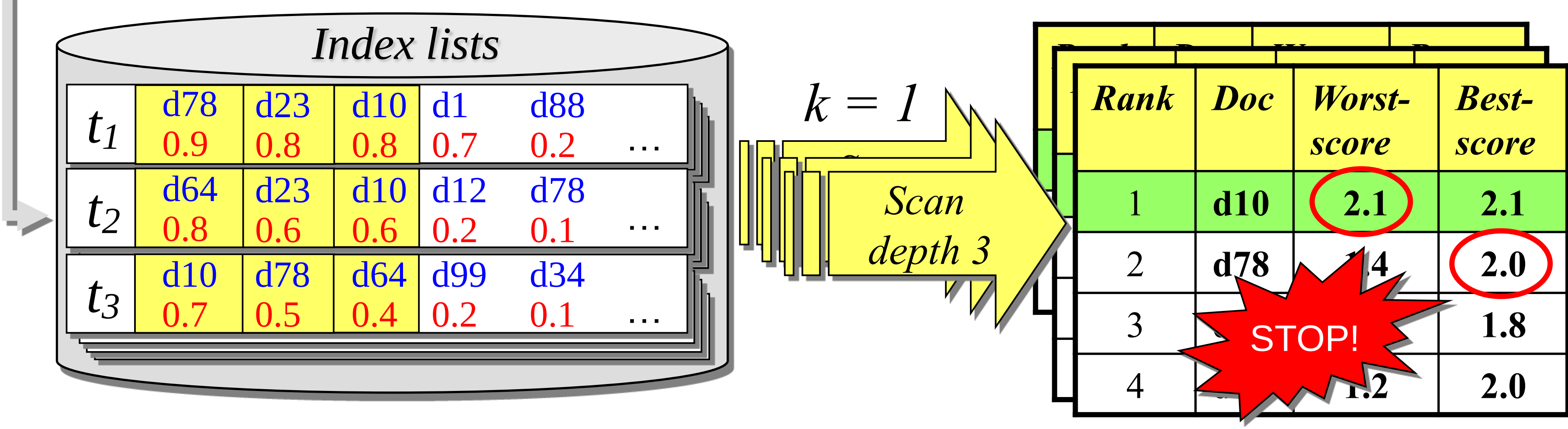
TA with Sorted Access only (NRA) [Fagin 01, Güntzer 01]

sequential access (SA) faster
than random access (RA)
by factor of 20-1000

Data items: $d_1, \ldots, d_n$



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



No-random-access algorithm (NRA):

scan index lists; consider d at pos_i in L_i;
$E(d) := E(d) \cup \{i\}$; $high_i := s(t_i,d)$;
worstscore(d) := $aggr\{s(t_\nu,d) \mid \nu \in E(d)\}$;
bestscore(d) := aggr{worstscore(d), $aggr\{high_\nu \mid \nu \notin E(d)\}\}$;
if worstscore(d) > min-k then add d to top-k
min-k := min{worstscore(d’) | d’ ∈ top-k};
else if bestscore(d) > min-k then
cand := cand ∪ {d};
threshold := max {bestscore(d’) | d’∈ cand};
if threshold ≤ min-k then exit;

TA Complexity and Instance Optimality

TA has worst-case run-time O($n^{\frac{m-1}{m}}$) with high prob. and space O(1)
NRA has worst-case run-time O(n) and space O(n)

Definition:
For class $\mathcal{A}$ of algorithms and class $\mathcal{D}$ of datasets,
algorithm B is **instance optimal** over $\mathcal{A}$ and $\mathcal{D}$ if
for every $A \in \mathcal{A}$ on $D \in \mathcal{D}$: cost(B,D) $\le$ c*O(cost(A,D)) + c'
($\rightarrow$ competitiveness c).

Theorem:
- **TA is instance optimal** over all algorithms that are based on sorted and random accesses to m lists (no „wild guesses").
- **NRA is instance optimal** over all algorithms with SA only.

if „wild guesses" are allowed,
then no deterministic algorithm is instance-optimal

Implementation Issues for TA Family

- Limitation of asymptotic complexity:
  - m (#lists) and k (#results) are important parameters
- Priority queues:
  - straightforward use of Fibonacci heap has high overhead
  - better: periodic rebuild of bounded-size PQs
- Memory management:
  - peak memory use as important for performance as scan depth
  - aim for early candidate pruning even if scan depth stays the same
- Hybrid block index:
  - pack index entries into big blocks in desc score order
  - keep blocks in score order
  - keep entries within a block in item id order
  - after each block read: merge-join first, then PQ update

Approximate Top-k Answers

- **IR heuristics** for impact-ordered lists [Anh/Moffat: SIGIR'01]:
  Accumulator Limiting, Accumulator Thresholding

- **Approximation TA** [Fagin et al.2003] :
  θ-approximation T' for q with $\theta > 1$ is a set T' of items with:
  - $|T'|=k$ and
  - for each $d' \in T'$ and each $d'' \notin T'$: $\theta * score(q,d') \geq score(q,d'')$

  Modified TA:
  ... stop when ***min-k*** $\boldsymbol{\geq aggr\ (high_1, ..., high_m) / \theta}$

- **Probabilistic Top-k** [Theobald et al. 2004] :
  guarantee small deviation from exact top-k result
  with high probability

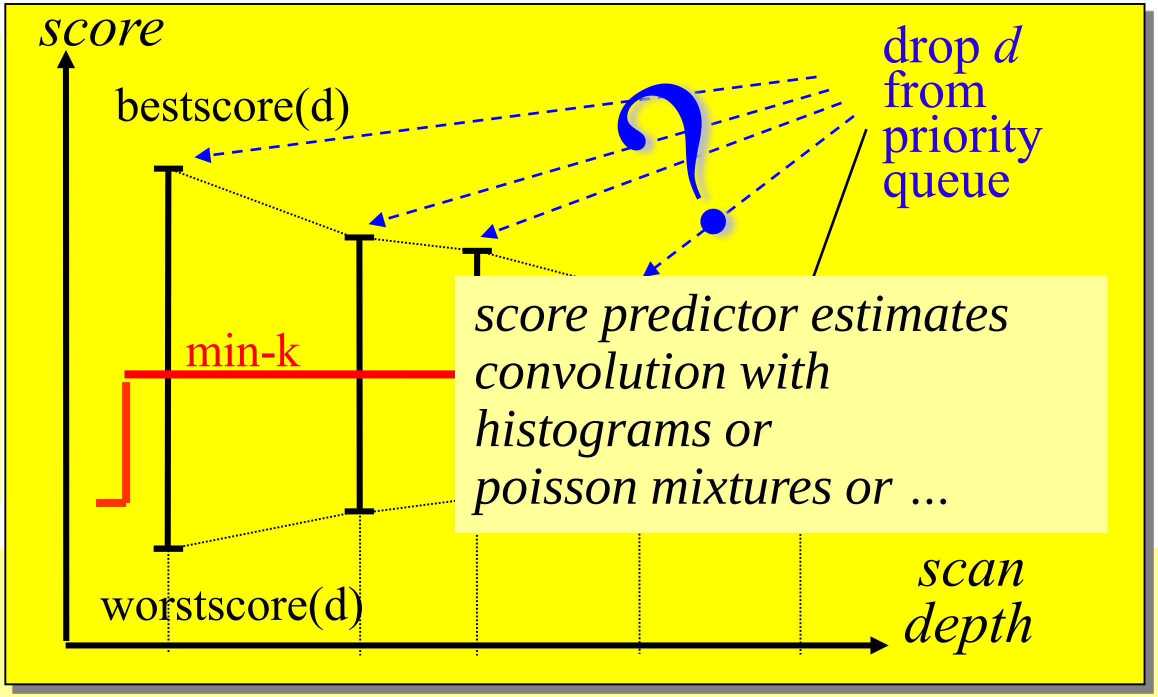
Probabilistic Top-k Answers

TA family of algorithms based on invariant (with sum as aggr):

$$\underbrace{\sum_{i \in E(d)} s_i(d)}_{\text{worstscore}(d)} \le s(d) \le \underbrace{\sum_{i \in E(d)} s_i(d) + \sum_{i \notin E(d)} high_i}_{\text{bestscore}(d)}$$

- Add d to top-k result, if *worstscore(d) > min-k*
- Drop d only if *bestscore(d) < min-k*, otherwise keep in PQ

→ Often overly conservative (deep scans, high memory for PQ)



→ Approximate top-k with probabilistic guarantees:

$$p(d) := P\left[\sum_{i \in E(d)} s_i(d) + \sum_{i \notin E(d)} S_i > \delta\right]$$ with δ = min-k

discard candidates d from queue if $p(d) \le \varepsilon$ $\Rightarrow$ E[rel. precision@k] = $1-\varepsilon$

Combined Algorithm (CA) for Balanced SA/RA Scheduling [Fagin et al. 03]

cost ratio $C_{RA}/C_{SA} = r$

perform NRA (TA-sorted)

...

after every r rounds of SA (m*r scan steps)

perform RA to look up all missing scores of „best candidate" in Q

cost competitiveness w.r.t. „optimal schedule"

(scan until $\Sigma_i\ high_i \le \min\{bestscore(d) \mid d \in$ final top-k$\}$,

then perform RAs for all d' with bestscore(d') > min-k): 4m + k

Flexible Scheduling of SA's and RA's for Top-k Query Processing

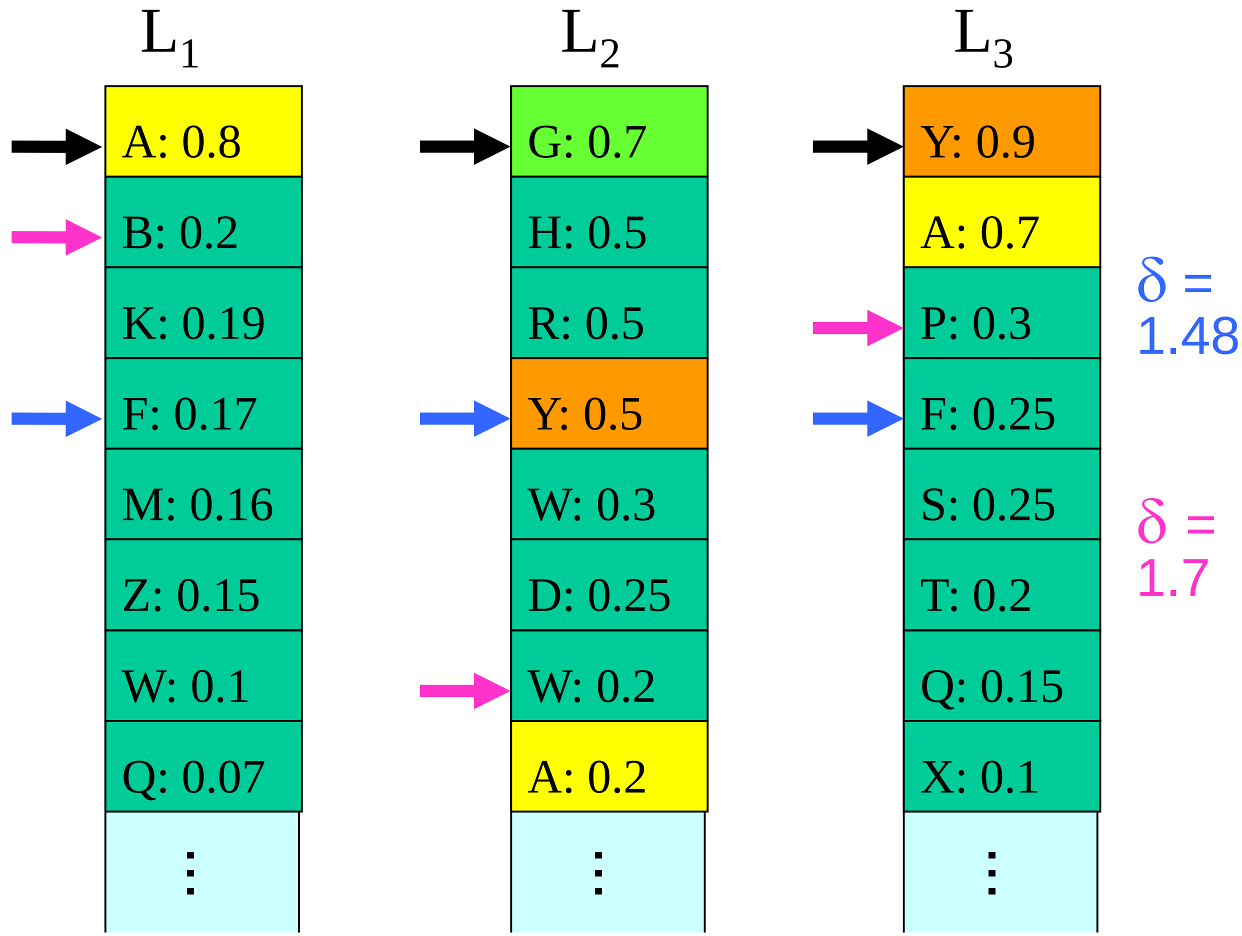
Goals:

1. decrease **$high_i$ upper-bounds** quickly
   → decreases bestscore for candidates
   → reduces candidate set
2. reduce **worstscore-bestscore gap** for most **promising candidates**
   → increases min-k threshold
   → more effective threshold test for other candidates

Ideas for better scheduling:

1. Non-uniform choice of SA steps in different lists
2. Careful choice of postponed RA steps for promising candidates when worstscore is high and worstscore-bestscore gap is small

Scheduling Example



batch of $b = \Sigma_{i=1..m}\, b_i$ steps:
choose b_i values so as to achieve high score reduction δ

+ carefully chosen RAs:
score lookups for „interesting“ candidates

Scheduling Example

L_1	L_2	L_3
A: 0.8	G: 0.7	Y: 0.9
B: 0.2	H: 0.5	A: 0.7
K: 0.19	R: 0.5	P: 0.3
F: 0.17	Y: 0.5	F: 0.25
M: 0.16	W: 0.3	S: 0.25
Z: 0.15	D: 0.25	T: 0.2
W: 0.1	W: 0.2	Q: 0.15
Q: 0.07	A: 0.2	X: 0.1
⋮	⋮	⋮

compute top-1 result
using flexible SAs and RAs

Scheduling Example

L_1	L_2	L_3
A: 0.8	G: 0.7	Y: 0.9
B: 0.2	H: 0.5	A: 0.7
K: 0.19	R: 0.5	P: 0.3
F: 0.17	Y: 0.5	F: 0.25
M: 0.16	W: 0.3	S: 0.25
Z: 0.15	D: 0.25	T: 0.2
W: 0.1	W: 0.2	Q: 0.15
Q: 0.07	A: 0.2	X: 0.1
⋮	⋮	⋮

candidates:

A: [0.8, 2.4]

G: [0.7, 2.4]

Y: [0.9, 2.4]

?: [0.0, 2.4]

Scheduling Example

L_1	L_2	L_3
A: 0.8	G: 0.7	Y: 0.9
B: 0.2	H: 0.5	A: 0.7
K: 0.19	R: 0.5	P: 0.3
F: 0.17	Y: 0.5	F: 0.25
M: 0.16	W: 0.3	S: 0.25
Z: 0.15	D: 0.25	T: 0.2
W: 0.1	W: 0.2	Q: 0.15
Q: 0.07	A: 0.2	X: 0.1
⋮	⋮	⋮

candidates:

A: [1.5, 2.0]

Y: [0.9, 1.6]

G: [0.7, 1.6]

~~?: [0.0, 1.4]~~

Scheduling Example

L_1	L_2	L_3
A: 0.8	G: 0.7	Y: 0.9
B: 0.2	H: 0.5	A: 0.7
K: 0.19	R: 0.5	P: 0.3
F: 0.17	Y: 0.5	F: 0.25
M: 0.16	W: 0.3	S: 0.25
Z: 0.15	D: 0.25	T: 0.2
W: 0.1	W: 0.2	Q: 0.15
Q: 0.07	A: 0.2	X: 0.1
⋮	⋮	⋮

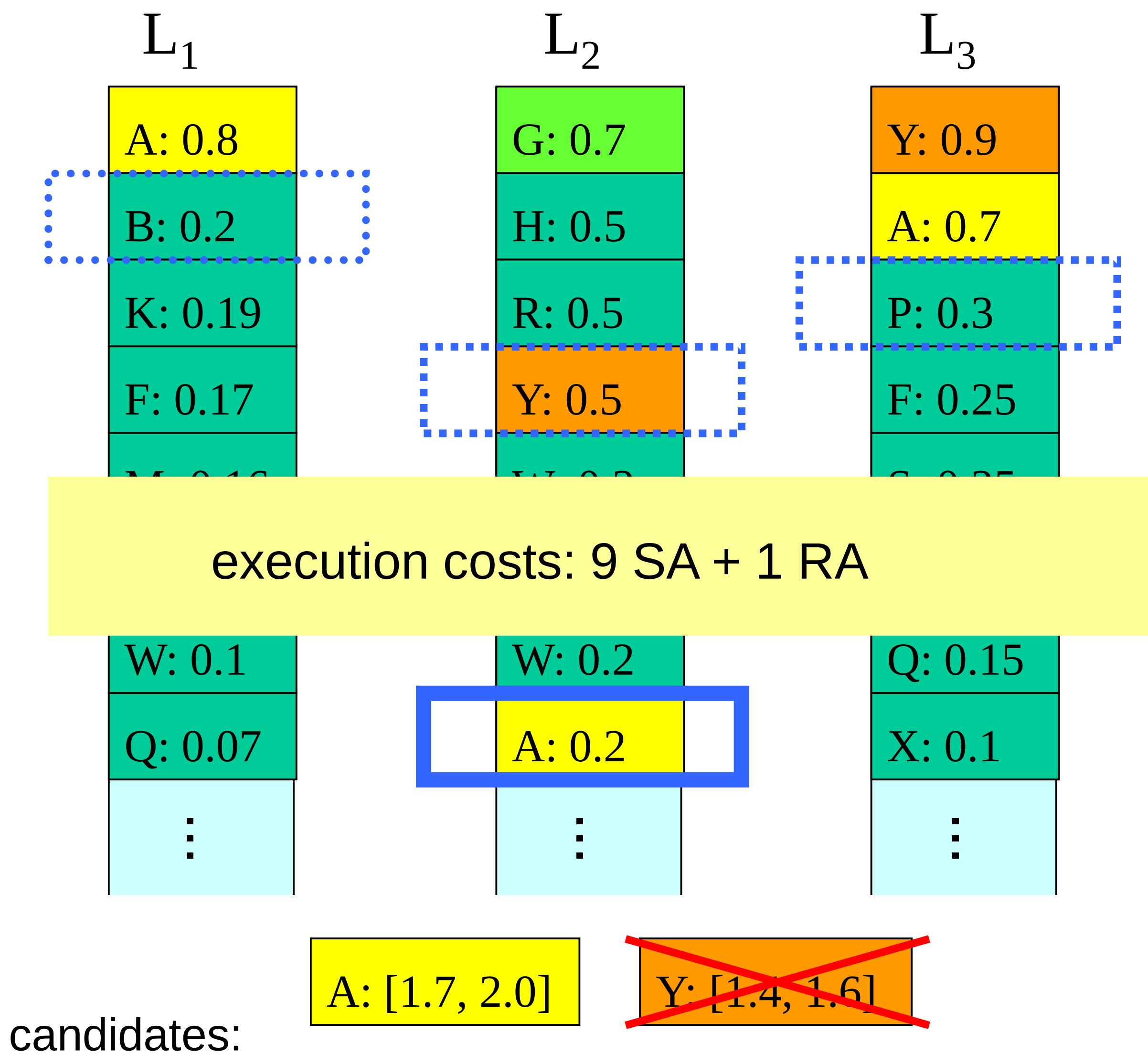
candidates:

A: [1.5, 2.0]

Y: [1.4, 1.6]

~~G: [0.7, 1.2]~~

Scheduling Example



Top-k Queries on Internet Sources

[Marian et al. 2004]

Setting:

- score-ordered lists **dynamically produced** by Internet sources
- some sources restricted to lookups only (no lists)

Example:

preference search for hotel based on ***distance, price, rating***
using ***mapquest.com, booking.com, tripadvisor.com***

Goal:

good **scheduling** for (parallel) access to restricted sources:
SA-sources, RA-sources, universal sources
with different costs for SA and RA

Method (Idea):

- scan all SA-sources in **parallel**
- in each step: choose next SA-source or perform RA on RA-source or universal source with best **benefit/cost** contribution

Top-k Rank Joins on Structured Data

[Ilyas et al. 2008]

extend TA/NRA/etc. to ranked query results from structured data (improve over baseline: evaluate query, then sort)

Select R.Name, C.Theater, C.Movie
From RestaurantsGuide R, CinemasProgram C
Where R.City = C.City
Order By R.Quality/R.Price + C.Rating Desc

RestaurantsGuide

Name	Type	Quality	Price	City
BlueDragon	Chinese	✶ ✶ ✶	€15	SB
Haiku	Japanese	✶ ✶ ✶ ✶	€30	SB
Mahatma	Indian	✶ ✶ ✶	€20	IGB
Mescal	Mexican	✶ ✶	€10	IGB
BigSchwenk	German	✶ ✶ ✶	€25	SLS
…				

CinemasProgram

Theater	Movie	Rating	City
BlueSmoke	Tombstone	7.5	SB
Oscar‘s	Hero	8.2	SB
Holly‘s	Die Hard	6.6	SB
GoodNight	Seven	7.7	IGB
BigHits	Godfather	9.1	IGB
…			

DAAT, TAAT, Top-k: Lessons Learned

- TA family over impact-ordered lists
  - is most elegant and potentially most efficient
  - but depending on score skew, it may degrade badly

- DAAT over document-ordered lists
  - is most versatile and robust
  - has lowest overhead and still allows pruning
  - can be easily scaled out on server farm

- TAAT is of interest for special use-cases
  (e.g. patent search with many keywords in queries)

12.3 Phrase Queries and Proximity Queries

phrase queries such as:

“Star Wars Episode 7“, “The Force Awakens“, “Obi Wan Kenobi“, “dark lord“
“Wir schaffen das“, “to be or not to be“, “roots of cubic polynomials“, “evil empire “

difficult to anticipate and index all (meaningful) phrases
sources could be thesauri/dictionaries or query logs

→ standard approach:
combine single-term index with separate position index

Inverted Index

term	doc	score
...		
empire	77	0.85
empire	39	0.82
...		
evil	49	0.81
evil	39	0.78
evil	12	0.75
...		
evil	77	0.12
...		

Position Index

term	doc	offset
...		
empire	39	191
empire	77	375
...		
evil	12	45
evil	39	190
evil	39	194
evil	49	190
...		
evil	77	190
...		

Bigram and Phrase Indexing

build index over all **word pairs (bigrams):**

index lists (term1, term2, doc, score) or

for each term1 nested list (term2, doc, score)

variations:

- treat nearest nouns as pairs,
  or discount articles, prepositions, conjunctions
- index **phrases from query logs**, compute correlation statistics

query processing by merging posting lists:

- decompose even-numbered phrases into bigrams
- decompose odd-numbered phrases into bigrams
  with low selectivity (as estimated by df(term1))
- may additionally use standard single-term index if necessary

Examples:

to be or not to be $\rightarrow$ (to be) (or not) (to be)

The Lord of the Rings $\rightarrow$ (The Lord) (Lord of) (the Rings)

Proximity Search

Example queries: ***root polynom three,***
high cholesterol measure, doctor degree defense

Idea: identify positions (pos) of all query-term occurrences
and reward short distances

keyword proximity score **[Büttcher/Clarke: SIGIR'06]:**
aggregation of per-term scores #
+ per-term-pair scores attributed to each term

$$score(t_1 \ldots t_m) = \sum_{i=1..m} \left(score(t_i) + \sum_{j \neq i} \left\{ \frac{idf(t_j)}{(pos(t_i) - pos(t_j))^2} \;\middle|\; \underbrace{\neg \exists t_k \, (pos(t_i) < pos(t_k) < pos(t_j) \text{ or} \ldots)}_{\text{count only pairs of query terms with no other query term in between}} \right\} \right)$$

cannot be precomputed
→ expensive at query-time

Example: Proximity Score Computation

It^{1} took^{2} the^{3} **sea^{4}** a^{5} thousand^{6} **years,^{7}**
A^{8} thousand^{9} **years^{10}** to^{11} trace^{12}
The^{13} granite^{14} features^{15} of^{16} this^{17} **cliff,^{18}**
In^{19} crag^{20} and^{21} scarp^{22} and^{23} base.^{24}

Query: {**sea**, **years**, **cliff**}

$$\mathrm{acc}(d,\mathrm{sea}) = \boxed{\frac{idf(\mathrm{years})}{(7-4)^2}}$$

$$\mathrm{acc}(d,\mathrm{years}) = \boxed{\frac{idf(\mathrm{sea})}{(7-4)^2}} + \boxed{\frac{idf(\mathrm{cliff})}{(18-10)^2}}$$

$$\mathrm{acc}(d,\mathrm{cliff}) = \boxed{\frac{idf(\mathrm{years})}{(18-10)^2}}$$

Efficient Proximity Search

Define aggregation function to be distributive [Broschart et al. 2007] rather than „holistic“ [Büttcher/Clarke 2006]:

precompute term-pair distances and sum up at query-time

$$\text{score}(t_1 \ldots t_m) = \sum\nolimits_{i=1..m} \Bigg(\text{score}(t_i) + \underbrace{\sum_{j \neq i} \left\{ \frac{idf(t_j)}{(p(t_i) - p(t_j))^2} \right\}}_{\text{count all pairs of query terms}} \Bigg)$$

result quality comparable to „holistic“ scores

index all pairs within max. window size
(or nested list of nearby terms for each term),
with precomputed pair-score mass

Ex.: Efficiently Computable Proximity Score

It^{1} took^{2} the^{3} **sea^{4}** a^{5} thousand^{6} **years,^{7}**
A^{8} thousand^{9} **years^{10}** to^{11} trace^{12}
The^{13} granite^{14} features^{15} of^{16} this^{17} **cliff,^{18}**
In^{19} crag^{20} and^{21} scarp^{22} and^{23} base.^{24}

Query: {**sea**, **years**, **cliff**}

$$acc(d, cliff, sea) = \boxed{\frac{1}{(18-4)^2}}$$

$$acc(d, cliff, years) = \boxed{\frac{1}{(18-7)^2}} + \boxed{\frac{1}{(18-10)^2}}$$

$$acc(d, sea, years) = \boxed{\frac{1}{(7-4)^2}} + \boxed{\frac{1}{(10-4)^2}}$$

Relevance Feedback

Given: a query q, a result set (or ranked list) D,
a user's assessment u: $D \rightarrow \{+, -\}$
yielding positive docs $D^+ \subseteq D$ and negative docs $D^- \subseteq D$

Goal: derive query q' that better captures the user's intention,
by adapting term weights in the query or by query expansion

Classical IR approach: ***Rocchio method*** (for term vectors)

$$q' = \alpha q + \frac{\beta}{|D^+|} \sum_{d \in D^+} d - \frac{\gamma}{|D^-|} \sum_{d \in D^-} d$$

with $\alpha, \beta, \gamma \in [0,1]$
and typically $\alpha > \beta > \gamma$

Modern approach: replace explicit feedback by **implicit feedback**
derived from **query&click logs** (pos. if clicked, neg. if skipped)

or rely on **pseudo-relevance feedback**:
assume that all top-k results are positive

Relevance Feedback using Text Classification or Clustering

Relevant and irrelevant docs (as indicated by user)
form two classes or clusters of text-doc-vector distribution

Classifier:

- train classifier on relevant docs as positive class
- run feature selection to identify best terms for expansion
- pass results of expanded query through classifier

Clustering:

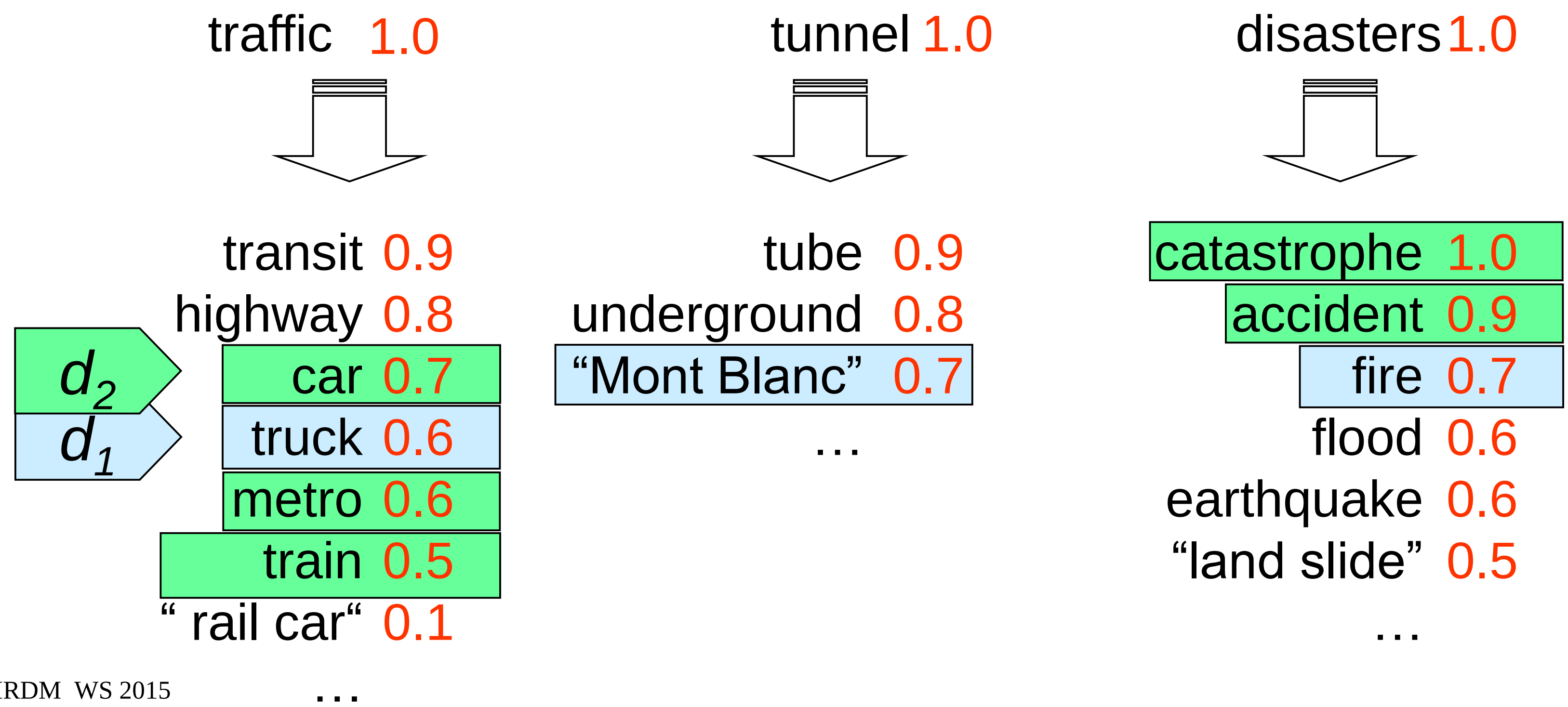
- refine clusters or compute sub-space clusters:
- user explores the resulting sub-clusters and guides expansion

Search engine examples:
http://exalead.com
http://yippy.com

Query Expansion

- Query expansion can be beneficial whenever high recall is needed
- **Expansion terms** can come from thesauri/dictionaries/ontologies or personalized profile, regardless of user feedback
- **Term-term similarities** precomputed from co-occurrence statistics

Example q: traffic tunnel disasters
(from TREC benchmark)



WordNet: Thesaurus/Ontology of Words and Concepts

http://wordnet.princeton.edu

200 000 concepts and lexical relations can be cast into
- **logical form or**
- **graph with weights for concept-concept relatedness strength**

WordNet Search - 3.1
- WordNet home page - Glossary - Help

Word to search for: traffic [Search WordNet]

word

Display Options: (Select option to change) [Change]

Key: "S:" = Show Synset (semantic) relations, "W:" = Show Word (

Display options for sense: (gloss) "an example sentence"

Noun

word sense (synset, Concept)

- S: (n) **traffic** (the aggregation of things (pedestrians or vehicles) coming and going in a particular locality during a specified period of time)
- S: (n) **traffic** (buying and selling; especially illicit trade)
- S: (n) **traffic** (the amount of activity over a communication system during a given period of time) *"heavy traffic overloaded the trunk lines"; "traffic on the internet is lightest during the night"*
- S: (n) dealings, **traffic** (social or verbal interchange (usually followed by `with'))

Verb

- S: (v) **traffic** (deal illegally) *"traffic drugs"*
- S: (v) **traffic** (trade or deal a commodity) *"They trafficked with us for gold"*

WordNet: Thesaurus/Ontology of Words and Concepts

Noun

- S: (n) **traffic** (the aggregation of things (pedestrians or vehicles) coming and going in a particular locality during a specified period of time)
    - *direct hyponym* / ***full hyponym***
        - S: (n) air traffic (traffic created by the movement of aircraft)
        - S: (n) commuter traffic (traffic created by people going to or returning from work)
        - S: (n) pedestrian traffic, foot traffic (people coming and going on foot)
        - S: (n) vehicular traffic, vehicle traffic (the aggregation of vehicles coming and going in a particular locality)
            - S: (n) automobile traffic, car traffic (cars coming and going)
            - S: (n) bicycle traffic (bicycles coming and going)
            - S: (n) bus traffic (buses coming and going)
            - S: (n) truck traffic (trucks coming and going)
    - *direct hypernym* / *inherited hypernym* / *sister term*
- S: (n) **traffic** (buying and selling; especially illicit trade)
- S: (n) **traffic** (the amount of activity over a communication system during a given period of time) *"heavy traffic overloaded the trunk lines"; "traffic on the internet is lightest during the night"*
- S: (n) dealings, **traffic** (social or verbal interchange (usually followed by `with'))

hyponyms (sub-concepts)

WordNet: Thesaurus/Ontology of Words and Concepts

Noun

- S: (n) **tunnel** (a passageway through or under something, usually underground (especially one for trains or cars)) *"the tunnel reduced congestion at that intersection"*

hyponyms (sub-concepts)

  - *direct hyponym* / *full hyponym*
    - S: (n) catacomb (an underground tunnel with recesses where bodies were buried (as in ancient Rome))
    - S: (n) railroad tunnel (a tunnel through which the railroad track runs)
    - S: (n) underpass, subway (an underground tunnel or passage enabling pedestrians to cross a road or railway)

meronyms (part-of)

  - *part meronym*
    - S: (n) shaft (a long vertical passage sunk into the earth, as for a mine or tunnel)
  - *domain category*
    - S: (n) car, auto, automobile, machine, motorcar (a motor vehicle with four wheels; usually propelled by an internal combustion engine) *"he needs a car to get to work"*

hypernyms (super-concepts)

  - ***direct hypernym*** / *inherited hypernym* / *sister term*
    - S: (n) passageway (a passage between rooms or between buildings)
  - *derivationally related form*
- S: (n) burrow, **tunnel** (a hole made by an animal, usually for shelter)

Robust Query Expansion

Threshold-based query expansion:

substitute w **by** $exp(w) := \{c_1 \ldots c_k\}$ ***with all*** c_i ***with*** $sim(w, c_i) \geq \delta$

Naive scoring:

$$s(q,d) = \Sigma_{w \in q} \Sigma_{c \in exp(w)} \; sim(w,c) * s_c(d)$$

risk of topic drift

Approach to careful expansion and scoring:

- determine **phrases** from query or best initial query results (e.g., forming 3-grams and looking up ontology/thesaurus entries)
- if **uniquely mapped** to one concept then expand with synonyms and weighted hyponyms
- avoid **undue score-mass accumulation** by expansion terms

$$s(q,d) = \Sigma_{w \in q} \; max_{c \in exp(w)} \{ sim(w,c) * s_c(d) \}$$

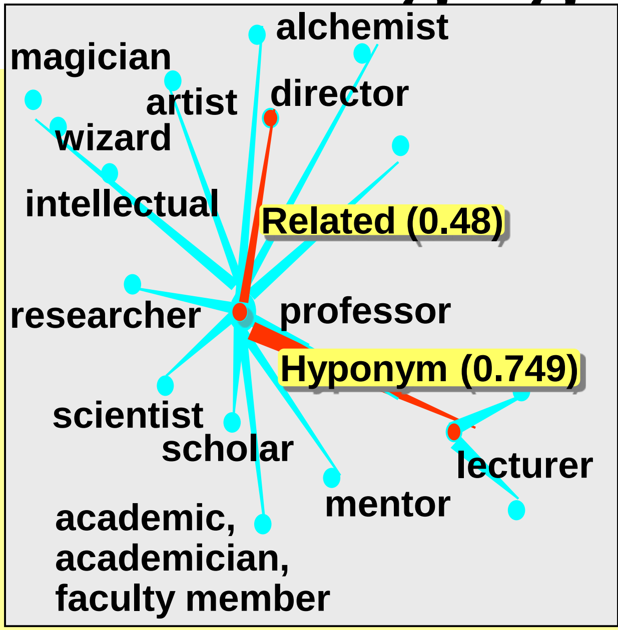
Query Expansion with Incremental Merging

[M. Theobald et al.: SIGIR 2005]

relaxable query q: *~professor research*
with expansions $exp(t)=\{w \mid sim(t,w) \geq \theta, t \in q\}$
based on ontology relatedness modulating
monotonic score aggregation by sim(t,w)

TA/NRA scans of index lists for $\bigcup_{t \in q} exp(t)$

Better: dynamic query expansion with
incremental merging of additional index lists



index on terms

research: 57: 0.6, 44: 0.4, 52: 0.4, 33: 0.3, 75: 0.3, ⋮

professor: 12: 0.9, 14: 0.8, 28: 0.6, 17: 0.55, 61: 0.5, 44: 0.5, ⋮

lecturer: 0.7: 37: 0.9, 44: 0.8, 22: 0.7, 23: 0.6, 51: 0.6, 52: 0.6, ⋮

scholar: 0.6: 92: 0.9, 67: 0.9, 52: 0.9, 44: 0.8, 55: 0.8, ⋮

meta-index (ontology / thesuarus)

professor:
lecturer: 0.7
scholar: 0.6
academic: 0.53
scientist: 0.5
...

efficient and robust

Query Expansion Example

From TREC 2004 Robust Track Benchmark:

Title: International Organized Crime

Description: Identify organizations that participate in international criminal activity, the activity, and, if possible, collaborating organizations and the countries involved.

Search Word: organized crime   Redisplay Overvie

Searches for organized crime: Noun   Senses:

1 sense of organized crime

Sense 1
organized crime, gangland, gangdom -- (underworld organizations)
=> yakuza -- (organized crime in Japan; an alliance of criminal organizations and illegal enterprises)
=> Mafia, Maffia, Sicilian Mafia -- (a secret terrorist group in Sicily; originally opposed tyranny but evolved into a criminal organization in the middle of the 19th century)
=> Black Hand -- (a secret terrorist society in the United States early in the 20th century)
=> Camorra -- (a secret society in Naples notorious for violence and blackmail)
=> syndicate, crime syndicate, mob, family -- (a loose affiliation of gangsters in charge of organized criminal activities)

Query Expansion Example

From TREC 2004 Robust Track Benchmark:

Title: **International Organized Crime**

Description: Identify organizations that participate in international criminal activity, the activity, and, if possible, collaborating organizations and the countries involved.

Query = {international[0.145|1.00],
~META[1.00|1.00][{gangdom[1.00|1.00], gangland[0.742|1.00],
"organ[0.213|1.00] & crime[0.312|1.00]", camorra[0.254|1.00], maffia[0.318|1.00],
mafia[0.154|1.00], "sicilian[0.201|1.00] & mafia[0.154|1.00]",
"black[0.066|1.00] & hand[0.053|1.00]", mob[0.123|1.00], syndicate[0.093|1.00]}],
organ[0.213|1.00], crime[0.312|1.00], collabor[0.415|0.20],
columbian[0.686|0.20], cartel[0.466|0.20], ...}}

135530 sorted accesses in 11.073s.

Results:

1. Interpol Chief on Fight Against Narcotics
2. Economic Counterintelligence Tasks Viewed
3. Dresden Conference Views Growth of Organized Crime in Europe
4. Report on Drug, Weapons Seizures in Southwest Border Region
5. SWITZERLAND CALLED SOFT ON CRIME

•••

Statistics for Term-Term Similarity or Concept-Concept Relatedness

Relatedness measures *sim(c1, c2)* based on WordNet-like thesaurus:

Wu-Palmer distance: |path(c1,lca(c1,c2))| + path(c2,lca(c1,c2))
with lowest common ancestor lca(c1,c2) in DAG|

Variants with edge weights based on edge type (hyponym, hypernym, …)

Relatedness measures *sim(c1, c2)* based on co-occurrences in corpus:

Dice coefficient:

$$\frac{2\left|\{docs\ with\ c1\}\cap\{docs\ with\ c2\}\right|}{\left|\{docs\ with\ c1\}\right| + \left|\{docs\ with\ c2\}\right|}$$

Jaccard coefficient:

$$\frac{\left|\{docs\ with\ c1\}\cap\{docs\ with\ c2\}\right|}{\left|\{docs\ with\ c1\}\right| + \left|\{docs\ with\ c2\}\right| - \left|\{docs\ with\ c1\ and\ c2\}\right|}$$

PMI (Pointwise Mutual Information):

$$\log \frac{freq(c1\ and\ c2)}{freq(c1)\cdot freq(c2)}$$

Conditional probability: $P[doc\ has\ c1 \mid doc\ has\ c2]$

Exploiting Query Logs for Query Expansion

Given: user sessions of the form (q, D+)
with clicked docs D+ (often only a single doc)

We are interested in the correlation between words
w in a query and w‘ in a clicked-on document:

$$P[w'|w] := P[w' \in d \;\; for \;\; some \;\; d \in D^+ \,|\, w \in q]$$

$$= \sum_{d \in D^+} \underbrace{P[w' \in d \,|\, d \in D^+]}_{\text{relative frequency of w‘ in d}} \cdot \underbrace{P[d \in D^+ \,|\, w \in q]}_{\text{relative frequency of d being clicked on when w appears in query}}$$

Estimate from query log:

Expand query by adding top m words w‘ in desc. order of $\prod_{w \in q} P[w'|w]$

Term-Term Similarity Estimation from Query-Click Logs

Use co-occurrences of

- term and term in **same query** (ordered terms)
- term in **query** and term in (title or URL of) **clicked doc**
- term in **query** without click and term in **next query**

to compute maximum-likelihood estimator for multinomial distribution for ordered term pairs or n-grams and derive **P[term u | term w] ~ freq[term u | term w]**

Useful for

- Suggestions for alternative queries (“did you mean …?“)
- Suggestions for auto-completion
- Background statistics for geo-localization or user-personalization

12.4 Query Result Diversification

True goal of search engine is to maximize

P[user clicks on at least one of the top-k results]

With **ambiguity of query terms** and **uncertainty about user intention**

(examples: “apple farm“, “mpi research“, “darwin expedition“, “Star Wars 7: The Force Awakens“, “physics nobel prize“, …)

we need to diversify the top-10 for risk minimization (portfolio mix)

Given a query q, query results $D=\{d_1, d_2, \ldots\}$,
similarity scores for results and the query $sim(d_i,q)$
and pair-wise similarities among results $sim(d_i,d_j)$

$\rightarrow$ Select top-k results $r_1, \ldots, r_k \in D$ such that

$$\alpha \sum_{i=1..k} sim(r_i, q) - (1-\alpha) \sum_{i \neq j} sim(r_i, r_j) = max!$$

Alternative Models for Diversification

Variant 1: Max-Min-Dispersion [Ravi, Rosenkrantz, Tayi 1994]
determine results set R={ $r_1, \ldots, r_k$ } such that

$$\alpha \min_{i=1..k} sim(r_i, q) - (1-\alpha) \max_{i \neq j} sim(r_i, r_j) = max!$$

Variant 2: intention-modulated [Agrawal et al. 2009]
assume that q may have m intentions $t_1..t_m$
(trained on query-click logs, Wikipedia disambiguation pages, etc.):
determine result set R with |R|=k such that

$$P[R\,|\,q] = \sum_{i=1}^{m} P[t_i\,|\,q] \cdot \underbrace{\left(1 - \prod_{r \in R} (1 - P[r\,|\,q, t_i])\right)}_{\text{at least one r clicked given intention } t_i \text{ for q}} = \max!$$

More variants in the literature, most are NP-hard
But many are **submodular** (have diminishing marginal returns)
→ **greedy algorithms** with approximation guarantees

Submodular Set Functions

Given a set Ω, a function f: $2^\Omega \rightarrow \Re$ is **submodular** if
for every X, Y $\subseteq \Omega$ with $X \subset Y$ and $z \in \Omega - Y$
the following **diminishing-returns property** holds
$$\mathbf{f(X \cup \{z\}) - f(X) \geq f(Y \cup \{z\}) - f(Y)}$$

Typical **optimization** problem aims to
choose a subset $X \subset \Omega$ that minimizes or maximizes f
under cardinality constraints for X

- these problems are usually NP-hard
  but often have polynomial algorithms with
  very good approximation guarantees
- greedy algorithms often yield very good
  approximate solutions

Maximal Marginal Relevance (MMR): Greedy Reordering for Diversification

[Carbonell/Goldstein 1998]

Compute a pool of top-m candidate results where m > k
(e.g. m=1000 for k=10)
Initialize S := ∅

Choose results in descending order of marginal utility:
repeat

$$S \coloneqq S \cup \; argmax_d \; (\alpha \; sim(d,q) - (1-\alpha) \sum_{r \in S} sim(r,d))$$

until |S|=k

Summary of Chapter 12

- **document-ordered** posting lists:
  QP based on scan and merge; can optimize order of lists
  and heuristically control memory for accumulators

- **impact-ordered** posting lists:
  top-k search can be sublinear with **Threshold Algorithm** family

- additional algorithmic options and optimizations for
  phrase and proximity queries and for **query expansion**

- with **ambiguity** of query terms and **uncertainty** of user intention,
  query result **diversification** is crucial

Additional Literature for Chapter 12

- J. Zobel, A. Moffat: Self-Indexing Inverted Files for Fast Text Retrieval, ACM TOIS 1996
- A. Broder, D. Carmel, M. Herscovici, A. Soffer, J. Zien: Efficient query evaluation using a two-level retrieval process, CIKM 2003
- R. Fagin, A. Lotem, M. Naor: Optimal Aggregation Algorithms for Middleware, Journal of Computer and System Sciences 2003
- C. Buckley, A. Lewit: Optimization of Inverted Vector Searches, SIGIR 1985
- I.F. Ilyas, G. Beskales, M.A. Soliman: A Survey of Top-k Query Processing Techniques in Relational Database Systems, ACM Comp. Surveys 40(4), 2008
- A. Marian, N. Bruno, L. Gravano: Evaluating Top-k Queries over Web-accessible Databases, ACM TODS 2004
- M. Theobald et al.: Top-k Query Processing with Probabilistic Guarantees, VLDB 2004
- H. Bast et al.: IO-Top-k: Index-access Optimized Top-k Query Processing. VLDB 2006
- H.E. Williams, J. Zobel, D. Bahle: Fast Phrase Querying with Combined Indexes, TOIS 2004
- A. Broschart, R. Schenkel: High-performance processing of text queries with tunable pruned term and term pair indexes. ACM TOIS 2012
- S. Büttcher, C. Clarke, B. Lushman: Term proximity scoring for ad-hoc retrieval on very large text collections. SIGIR 2006
- R. Schenkel et al.: Efficient Text Proximity Search, SPIRE 2007

Additional Literature for Chapter 12

- G. Miller, C. Fellbaum: WordNet: An Electronic Lexical Database. MIT Press 1998
- B. Billerbeck, F. Scholer, H.E. Williams, J. Zobel: Query expansion using associated queries. CIKM 2003
- S. Liu, F. Liu, C.T. Yu, W. Meng: An effective approach to document retrieval via utilizing WordNet and recognizing phrases, SIGIR 2004
- M. Theobald, R. Schenkel, G. Weikum: Efficient and Self-Tuning Incremental Query Expansion for Top-k Query Processing, SIGIR 2005
- H. Bast, I. Weber: Type Less, Find More: Fast Autocompletion Search with a Succinct Index, SIGIR 2006
- Z. Bar-Yossef, M. Gurevich: Mining Search Engine Query Logs via Suggestion Sampling. PVLDB 2008
- Z. Bar-Yossef, N. Kraus: Context-sensitive query auto-completion. WWW 2011
- T. Joachims et al.: Evaluating the accuracy of implicit feedback from clicks and query reformulations in Web search, TOIS 2007
- SP.Chirita, C.Firan, W.Nejdl: Personalized Query Expansion for the Web. WWW 2007
- J. Carbonell, J. Goldstein: The Use of MMR: Diversity-based Reranking, SIGIR 1998
- R. Agrawal, S. Gollapudi, A. Halverson, S. Ieong: Diversifying search results. WSDM 2009
- S. Gollapudi, A. Sharma: An Axiomatic Approach for Result Diversification, WWW 2009