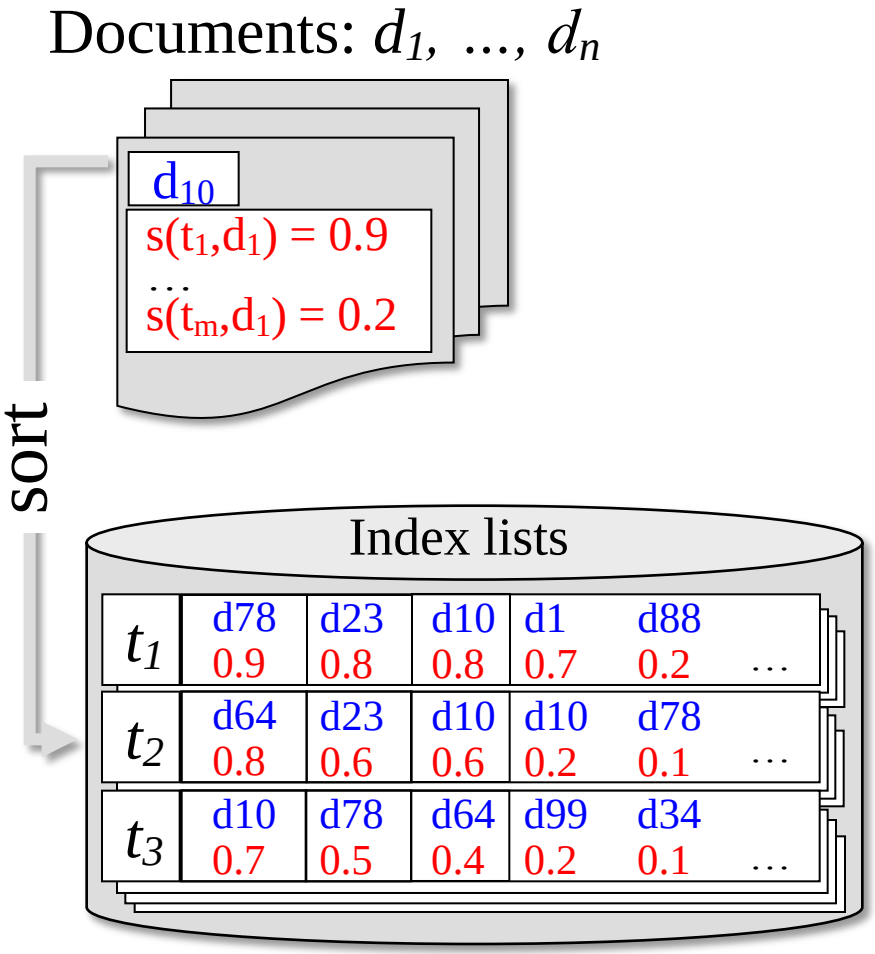
V.3 Top-k Query Processing

3.1 IR-style heuristics for efficient inverted index scans

3.2 Fagin's family of threshold algorithms (TA)

3.3 Approximation algorithms based on TA

Indexing with Score-ordered Lists



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

Aims to avoid having to read entire lists:
→ rather scan only (short) prefixes of lists for the **top-ranked answers**.

Query Processing on Score-ordered Lists

Top-k aggregation query over R(docId, A_1, ..., A_m) partitions:

Select docId, score($R_1.A_1$, ..., $R_m.A_m$) As Aggr
From Outer Join R_1, …, R_m
Order By Aggr Desc Limit k

with **monotone score aggregation function** score:

score: $R^m \rightarrow R$, s.t. $(\forall\ x_i \geq x_i') \Rightarrow \text{score}(x_1 \ldots x_m) \geq \text{score}(x_1' \ldots x_m')$

- Precompute index **lists in descending attr-value order** (score-ordered, impact-ordered).
- Scan lists by **sorted access** (SA) in round-robin manner.
- Perform **random accesses** (RA) by docId when convenient.
- Compute aggregation score **incrementally** in **candidate queue**.
- Compute **score bounds** for candidate results and stop when **threshold test** guarantees correct top-k (or when heuristics indicate "good enough" approximation).

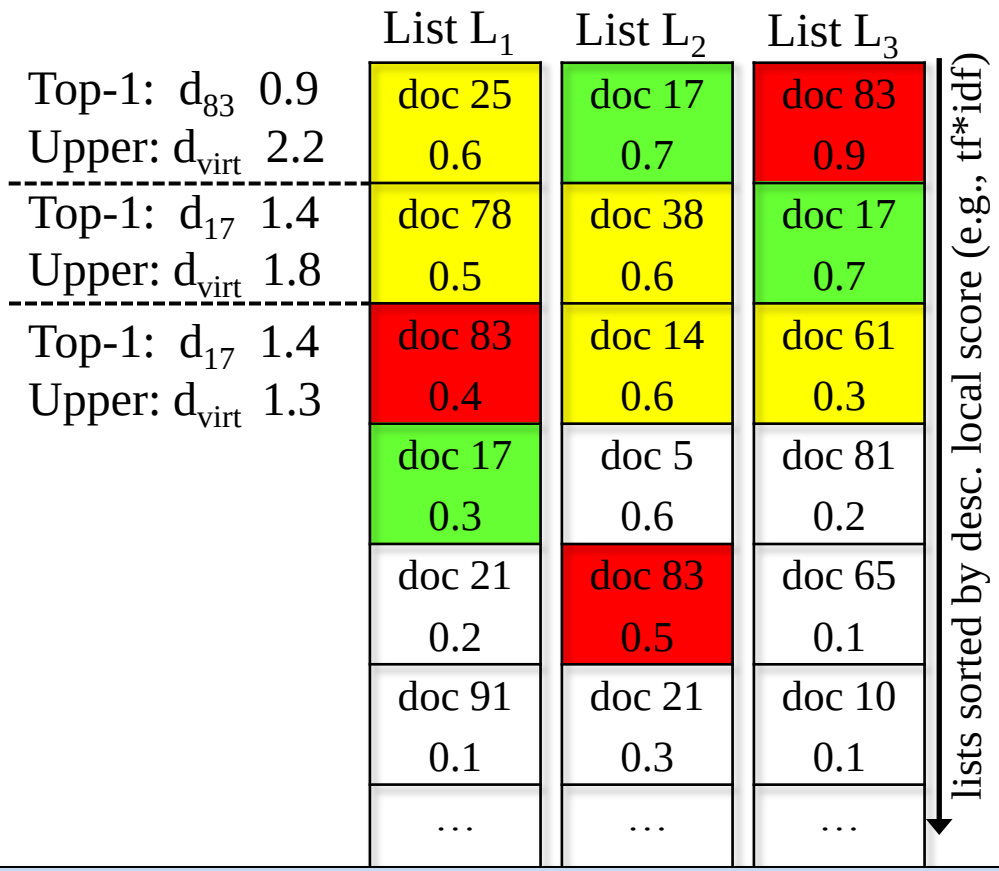
V.3.1 Heuristic Top-k Algorithms

General pruning and index-access ordering heuristics:

- Disregard index lists with **low idf** (below given threshold).
- For scheduling index scans, give **priority** to index lists that are short and have **high idf**.
- Stop adding candidates to the queue if we **run out of memory**.
- Stop scanning a particular list if the **local scores** in it become **low**.
- …

Buckley'85

[Buckley & Lewit: SIGIR'85]



1) Incrementally scan lists L_i in round-robin fashion.
2) For each access, aggregate local score to corresponding document's global score.
3) The sum of local scores at the current scan positions is an ***upper bound*** for all unseen documents ("virtual doc").
4) Stop if this upper bound is less than current k-th best document's partial score.

Note: this is a simplified version of Buckley's original algorithm, which considers an upper bound for the actual (k+1)-ranked document instead of the virtual document.
If this (k+1)-ranked document is computed properly (e.g., all candidates are kept and updated in a queue), then this is the first correct top-k algorithm based on sequential data access proposed in the literature!

Quit & Continue

[Moffat/Zobel: TOIS'96]

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

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

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

***quit* heuristics:** (with lists ordered by tf or tf*idl):

- Ignore index list L_i if idf(t_i) is below threshold.
- Stop scanning L_i if tf(t_i,d_j)*idf(t_i)*idl(d_j) drops below threshold.
- Stop scanning L_i when the number of accumulators is too high.

***continue* heuristics:**

Upon reaching threshold, continue scanning index lists and aggregate scores but do not add any new documents to the accumulators.

Greedy Index Access Scheduling (I)

Assume index lists are sorted by descending $s_i(t_i,d_j)$ (e.g., using $tf(t_i,d_j)$ or $tf(t_i,d_j)*idl(d_j)$ values):

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 $\mathbf{s_i(t_i, pos(i))}$;

Update the accumulator of the corresponding doc at pos(g);

Increment pos(g);

Until stopping condition holds;

Greedy Index Access Scheduling (II)

[Güntzer, Balke, Kießling: "Stream-Combine", ITCC'01]

Assume index lists are sorted by descending $s_i(t_i,d_j)$:

Open scan cursors on all m index lists L(i);

Repeat

For sliding window w (e.g., 100 steps), find pos(g) among

current cursor positions pos(i) (i=1..m)

with the largest gradient $\mathbf{(s_i(t_i, pos(i)–w) – s_i(t_i, pos(i)))/w}$;

Update the accumulator of the corresponding doc at pos(g);

Increment pos(g);

Until stopping condition holds;

QP with Authority/Similarity Scoring

[Long/Suel: VLDB'03]

Focus on **score(q,d_j) = r(d_j) + s(q,d_j)**

with normalization r(·) ≤ a, s(·) ≤ b (and often a+b=1)

Keep index lists sorted in descending order of **"static" authority r(d_j)**

Conservative authority-based pruning:

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

→ Effective when total score of top-k results is dominated by r

***First-k' heuristics*:**

Scan all m index lists until k' ≥ k docs have been found
that appear in all lists.

→ This stopping condition is easy to check because lists are sorted by r.

Top-k with "Champion Lists"

Idea (Brin/Page'98):
In addition to the full index lists L_i sorted by r,
keep short ***"champion lists"*** (aka. "fancy lists") F_i that contain
docs d_j with the **highest values of $s_i(t_i,d_j)$** and **sort these lists by r.**

Champions First-k' heuristics:

Compute total score for all docs in $\cap F_i$ (i=1..m)
    and keep top-k results;
Cand := $\cup_i F_i - \cap_i F_i$;
For each $d_j \in$ Cand do {compute partial score of d_j};
Scan full index lists L_i (i=1..k);
    if pos(i) $\in$ Cand
        {add $s_i(t_i,pos(i))$ to partial score of doc at pos(i)}
    else {add pos(i) to Cand and set its partial score to $s_i(t_i,pos(i))$};
    Terminate the scan when we have k' docs with complete total score;

V.3.2 Fagin's Family of Threshold Algorithms

Threshold Algorithm (TA)

- Original version, often used as synonym for entire family of top-k algorithms.
- But: eager random access to candidate objects required.
- Worst-case memory consumption is strictly bounded → O(k)

No-Random-Access Algorithm (NRA)

- No random access required at all, but may have to scan large parts of the index lists.
- Worst-case memory consumption bounded by index size → O(m*n + k)

Combined Algorithm (CA)

- Cost-model for scheduling well-targeted random accesses to candidate objects.
- Algorithmic skeleton very similar to NRA, but typically terminates much faster.
- Worst-case memory consumption bounded by index size → O(m*n + k)

Different variants of TA family have been developed by several groups at around the same time.

Solid theoretical foundation (including proofs of instance optimality) provided in:
[R. Fagin, A. Lotem, M. Naor: Optimal Aggregation Algorithms for Middleware, JCSS'03]

Implementation (e.g., queue management) not specified by Fagin's framework (but does matter a lot in practice).

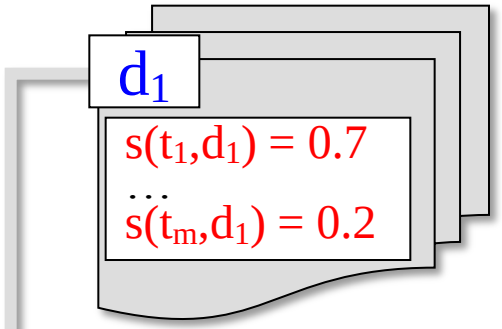
Many extensions for approximate variants of TA.

Threshold Algorithm (TA)

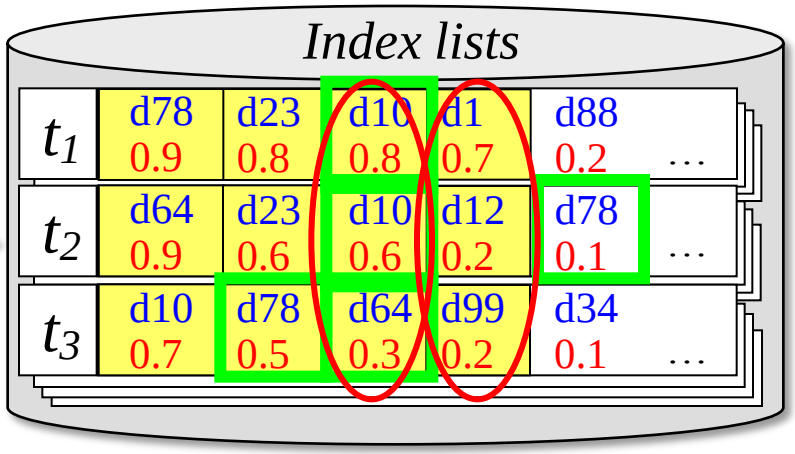
[Fagin'01, Güntzer'00, Nepal'99, Buckley'85]

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

Documents: $d_1, \ldots, d_n$

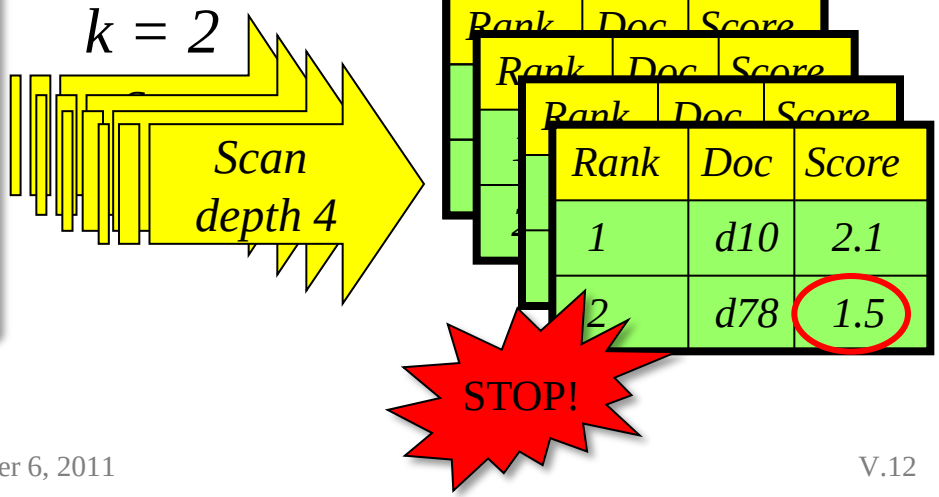


Query: $q = (t_1, t_2, t_3)$



Threshold algorithm (TA):

scan index lists; consider d at pos_i in L_i;
$high_i := s(t_i,d)$;
if d ∉ top-k then {
  look up $s_v(d)$ in all lists L_v with $v \neq i$;
  score(d) := aggr $\{s_v(d) \mid v=1..m\}$;
if score(d) > min-k then
  add d to top-k and remove min-score d';
  min-k := min{score(d') | d' ∈ top-k};
threshold := aggr $\{high_v \mid v=1..m\}$;
if threshold ≤ min-k then exit;

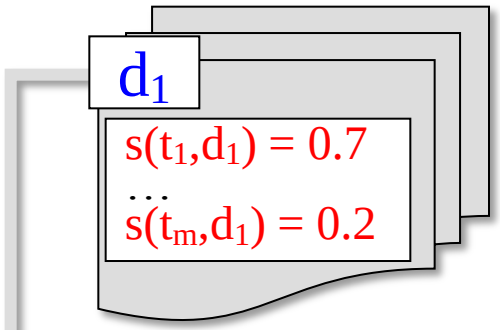


No-Random-Access Algorithm (NRA)

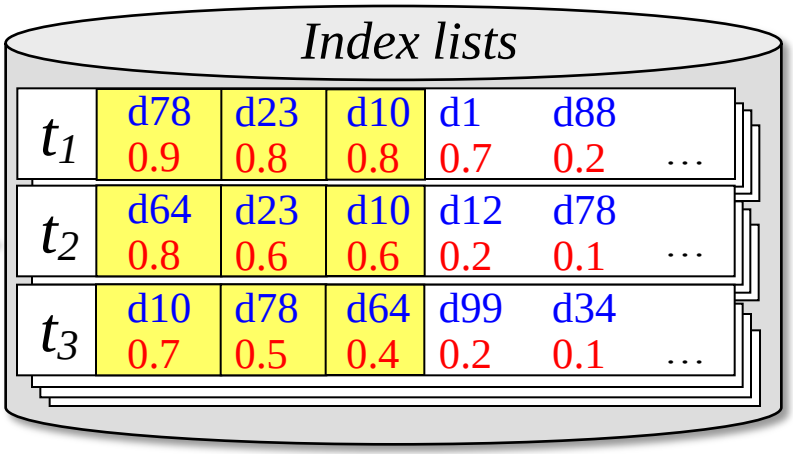
[Fagin'01, Güntzer'01]

Sequential access (SA) faster than random access (RA)

Documents: $d_1, \ldots, d_n$

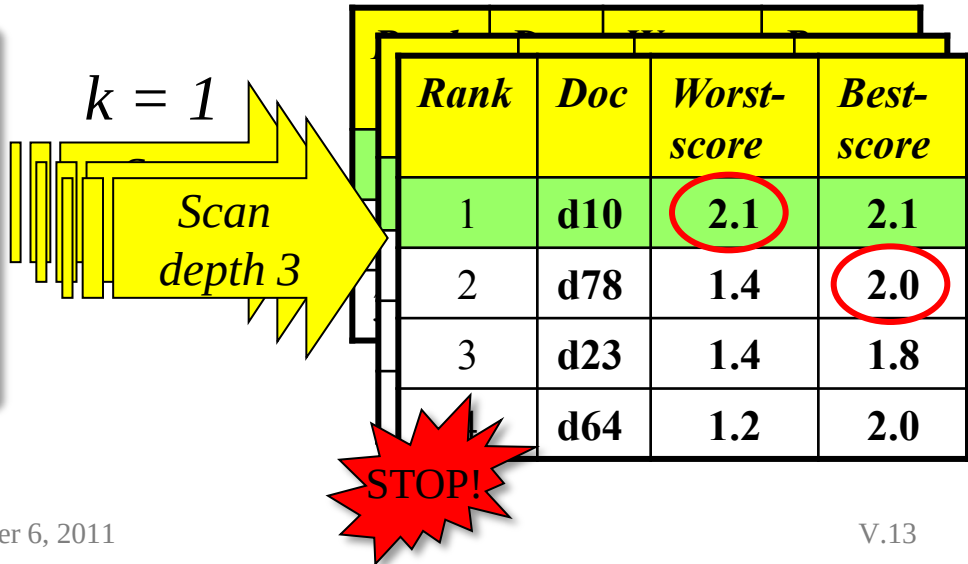


Query: $q = (t_1, t_2, t_3)$



<u>No-Random-Access algorithm (NRA):</u>

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 := $min\{worstscore(d') \mid d' \in$ top-k$\}$;
else if bestscore(d) > min-k then
    cand := cand $\cup \{d\}$;
threshold := $max\{bestscore(d') \mid d' \in cand\}$;
if threshold $\leq$ min-k then exit;



Combined Algorithm (CA)

[Fagin et al. '03]

Balanced SA/RA Scheduling:

Define cost ratio $\mathbf{C_{RA}/C_{SA} =: r}$

(e.g., based on statistics for execution environment ("middleware"), typical values: $C_{RA}/C_{SA} \sim$ 20-10,000 for a hard-disk)

Perform **NRA** (using sorted access only)

...

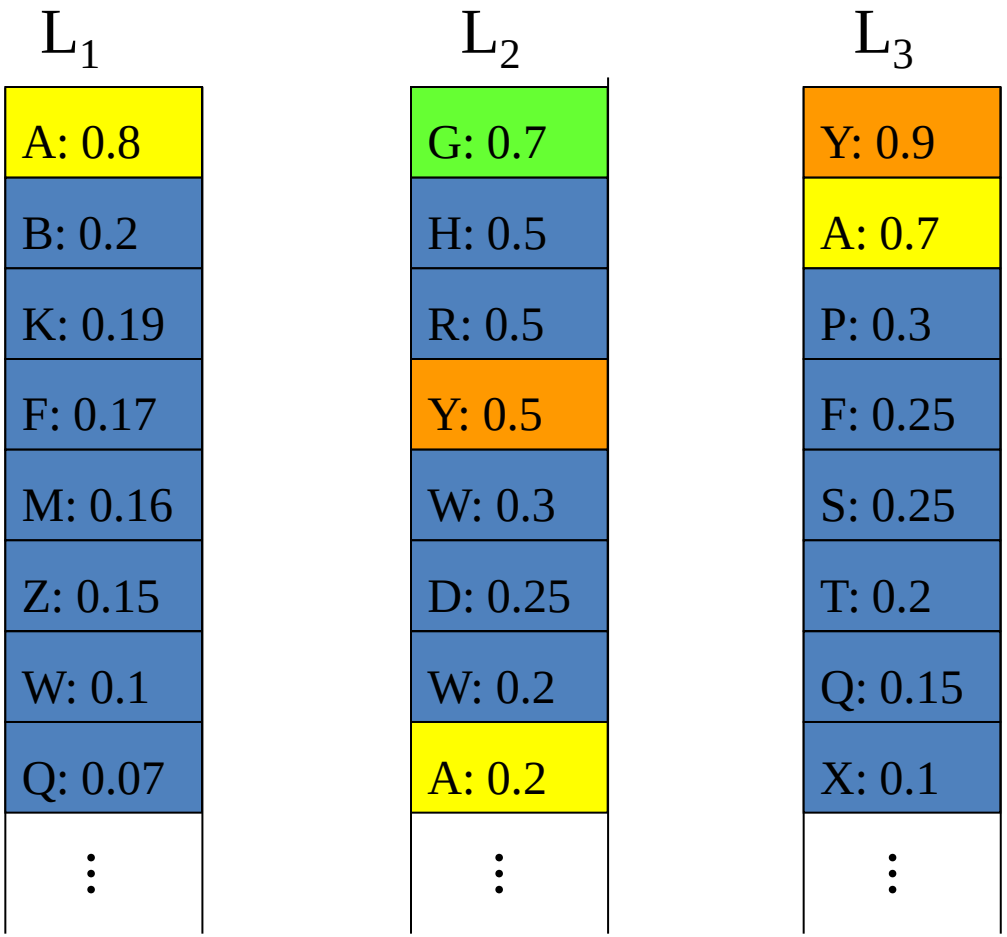
After every **r rounds of SA** (i.e., after m*r SA steps) perform **one RA** to look up the unknown scores of the **best candidate d** (w.r.t worstscore(d)) that is not among the current top-k items.

Cost **competitiveness** w.r.t. "optimal schedule"

(scan until $\Sigma_i \text{high}_i \leq \min\{\text{bestscore(d)} \mid d \in \textbf{final top-k}\}$,

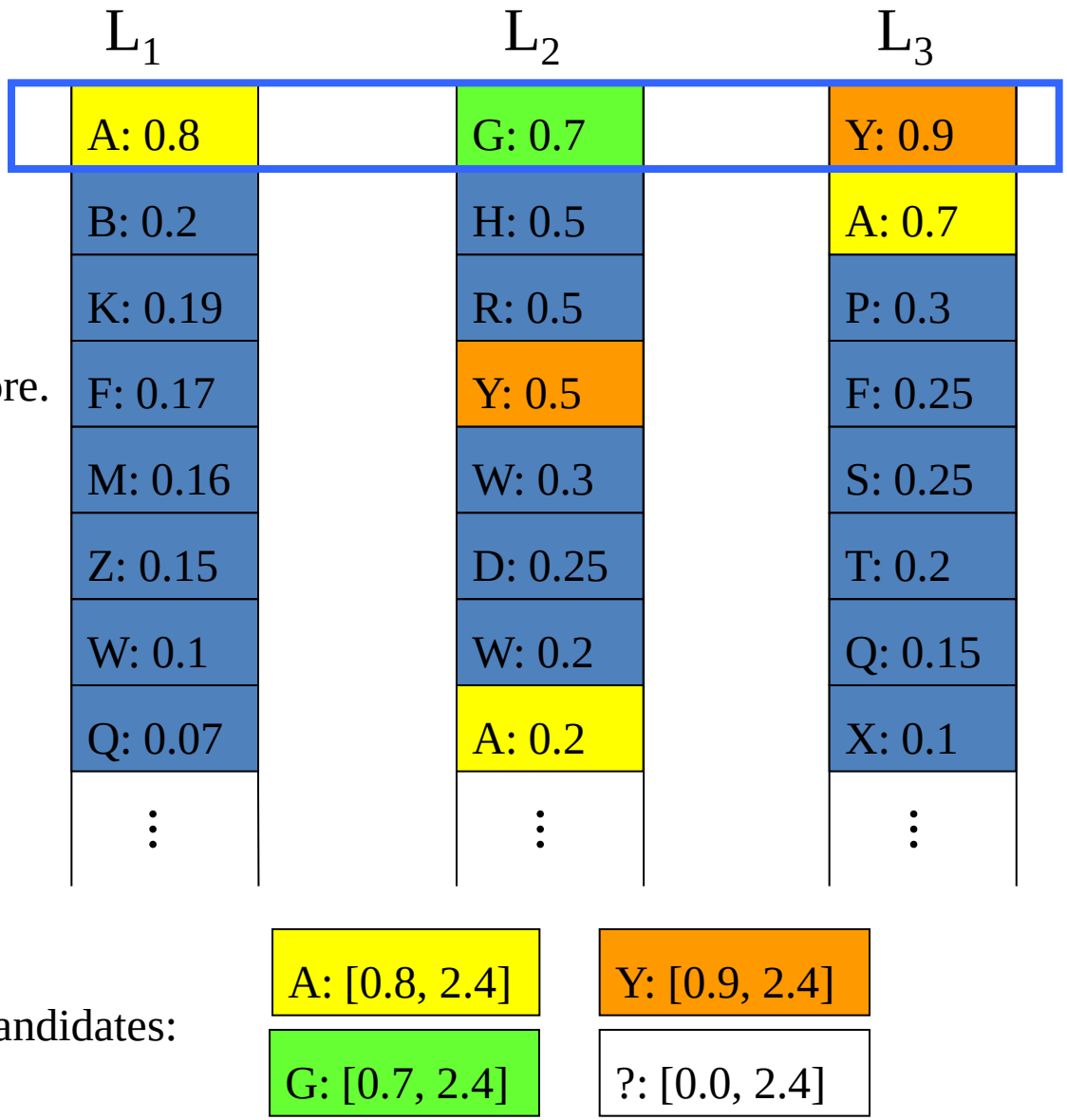
then perform RAs for all d' with bestscore(d') > min-k): **4m + k**

CA Scheduling Example



CA: compute top-1 result
using one RA after every round of SA

CA Scheduling Example



1st round of SA:

Y is top-1 w.r.t. worstscore.

A is best candidate w.r.t. worstscore.

→ Schedule RA for all of A's missing scores.

candidates:

A: [0.8, 2.4]

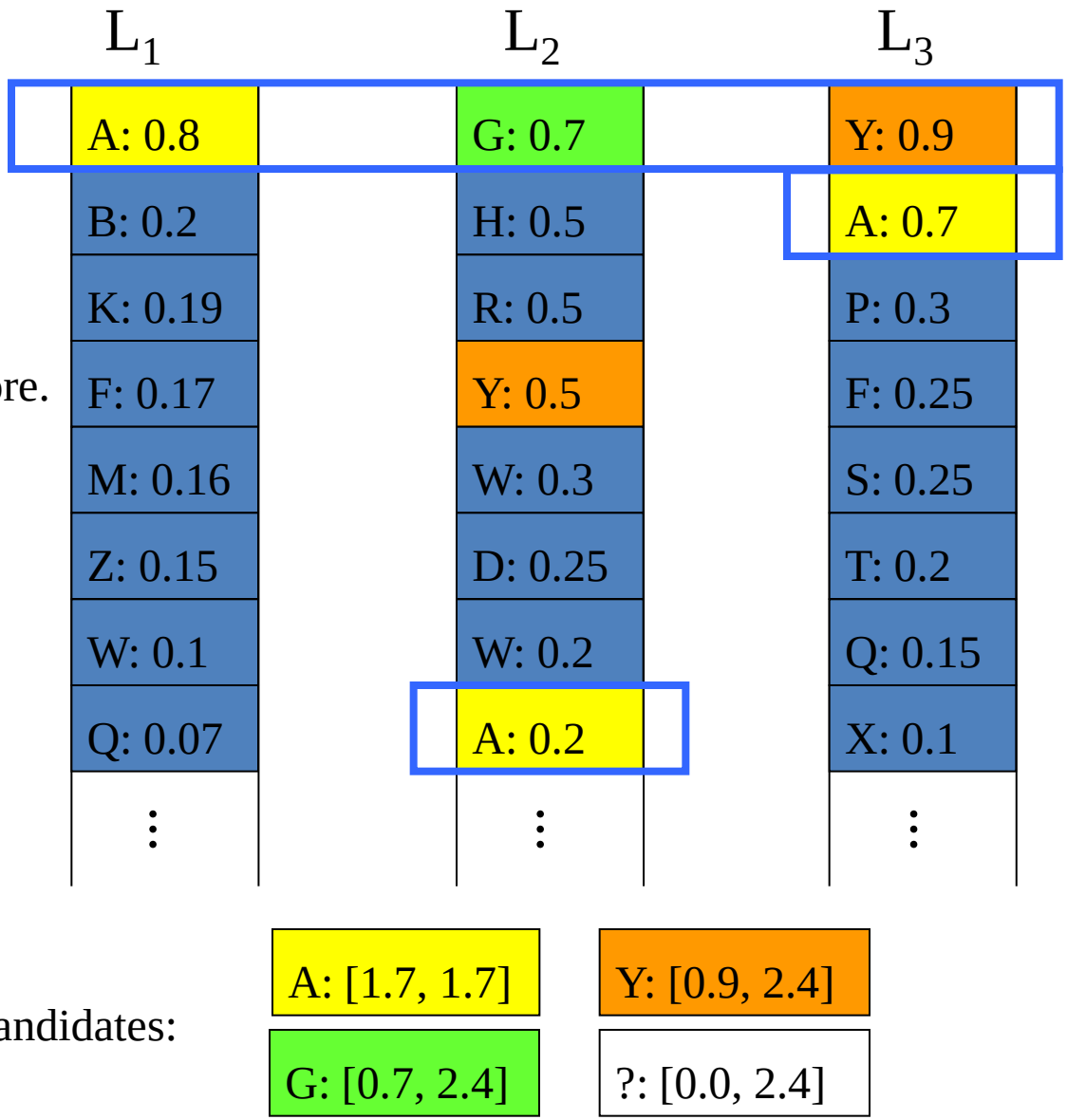
Y: [0.9, 2.4]

G: [0.7, 2.4]

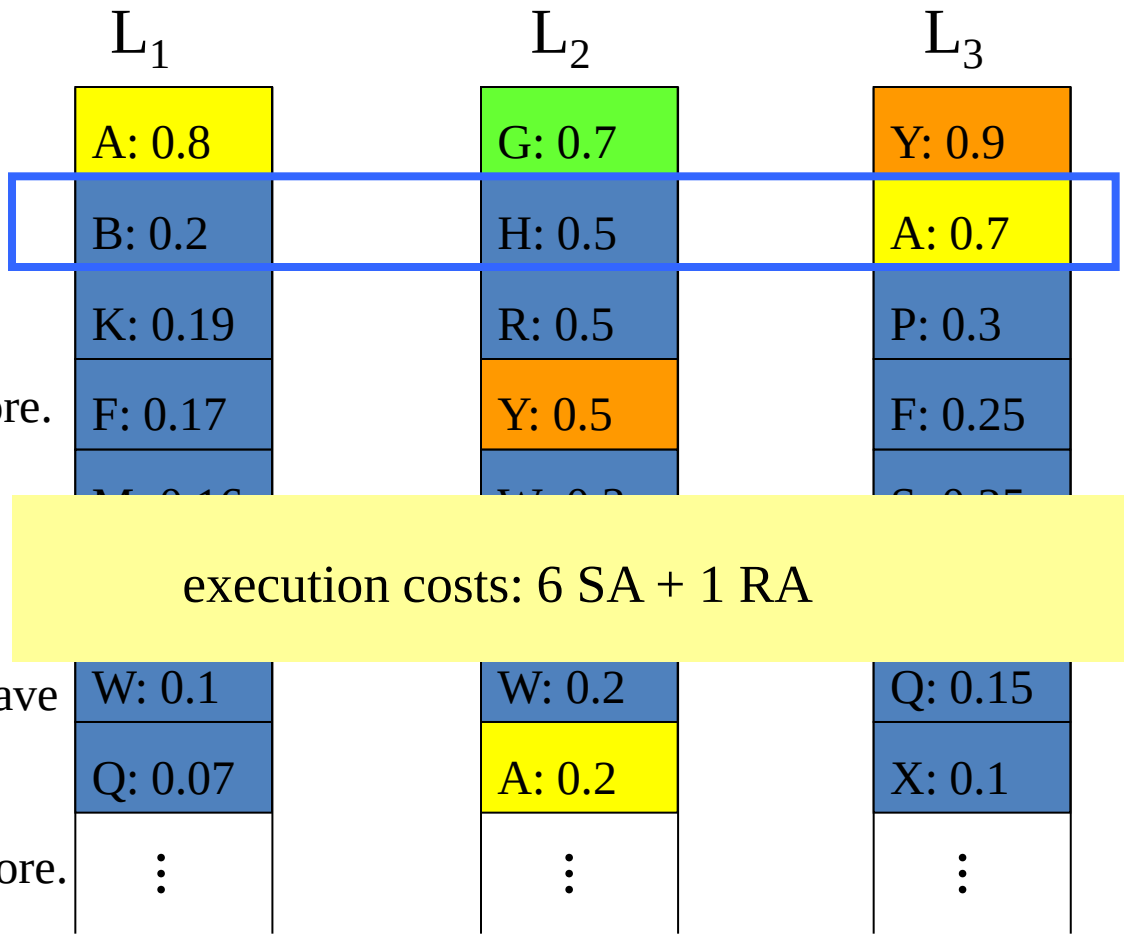
?: [0.0, 2.4]

CA Scheduling Example

1st round of SA:
Y is top-1 w.r.t. worstscore.
A is best candidate w.r.t. worstscore.
→ Schedule RA for all of A's missing scores.



CA Scheduling Example

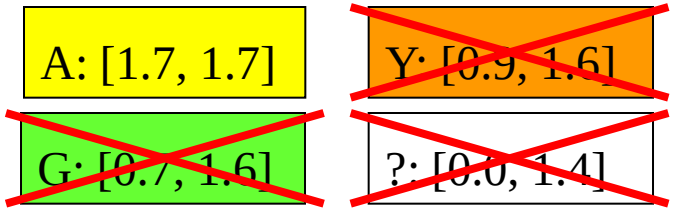


1st round of SA:
Y is top-1 w.r.t. worstscore.
A is best candidate w.r.t. worstscore.
→ Schedule RA for all of A's missing scores.

execution costs: 6 SA + 1 RA

2nd round of SA:
A is top-1 (worst- and bestscore have converged).
All candidate's (incl. virtual doc) bestscores are below A's worstscore.
→ Done!

candidates:



A note on counting SA's vs. RA's:

Although A is looked up in both L_2 and L_3 via random access in the previous example, this step is typically counted as just a single RA (as this can be implemented by one index lookup over a corresponding index structure that points to all the per-term scores of a document).

For SA's, we count each lookup of a document in one of the index lists as a separate SA. That is, one iteration of SA's over m lists in round-robin fashion yields m SA's.

Overall, counting the cost of SA's vs. RA's is highly implementation-dependent and can also be reflected in the C_{RA}/C_{SA} cost ratio considered by some of these algorithms.

For simplicity, we will use the convention above.

TA / NRA / CA Instance Optimality

[Fagin et al.'03]

Definition:

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

Theorem:

- **TA is instance optimal** over all top-k algorithms based on sorted and random accesses to m lists (no "wild guesses") and given cost ratio C_{RA}/C_{SA}.
- **NRA is instance optimal** over all algorithms with SA only.
- **CA is instance optimal** over all algorithms with SA and RA and given cost ratio C_{RA}/C_{SA}.

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

Top-k with Boolean Search Conditions

Combination of AND and ANDish: $(t_1$ AND … AND $t_j)$ t_{j+1} t_{j+2}… t_m

- TA family applicable with mandatory probing in AND lists
  → RA scheduling
- (worstscore, bestscore) bookkeeping and pruning
  more effective with "boosting weights" for AND lists

Combination of AND, "andish", and NOT:

NOT terms considered k.o. criteria for results
TA family applicable with mandatory probing for AND and NOT
→ RA scheduling for "expensive predicates"

Combination of AND, OR, NOT in Boolean sense:

- Best processed by index lists in docId order (not top-k!)
- Construct operator tree and push selective operators down;
  needs good query optimizer (selectivity estimation)

Implementation Issues for TA Family

- Limitation of asymptotic complexity:
  - **m** (#lists) and **k** (#results) are important **parameters**
- Priority queues:
  - Straightforward use of a heap (even for Fibonacci) has **high overhead**
  - Better: **periodic rebuild** of queue with partial sort O(n log k)
- 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 descending score order
  - Keep **blocks in score order**
  - Keep **entries within a block in docId order**
  - After each block read: merge-join first, then PQ update

V.3.3 Approximation Algorithms based on TA

- ***IR heuristics*** for impact-ordered lists [Anh/Moffat: SIGIR'01]:
  Accumulator limiting, accumulator thresholding, etc.

- **Approximation TA** [Fagin et al.: JCSS'03]:
  θ-approximation (approx. top-k results) T' for query q with $\theta > 1$ is a set T' of docs 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 $\geq$ aggr $(high_1, ..., high_m) / \theta$

- **Probabilistic Top-k** [Theobald et al.: VLDB'04]:
  Guarantee only small deviation from exact top-k result with high probability!

Top-k with Probabilistic Guarantees

[M. Theobald et al.: VLDB'04]

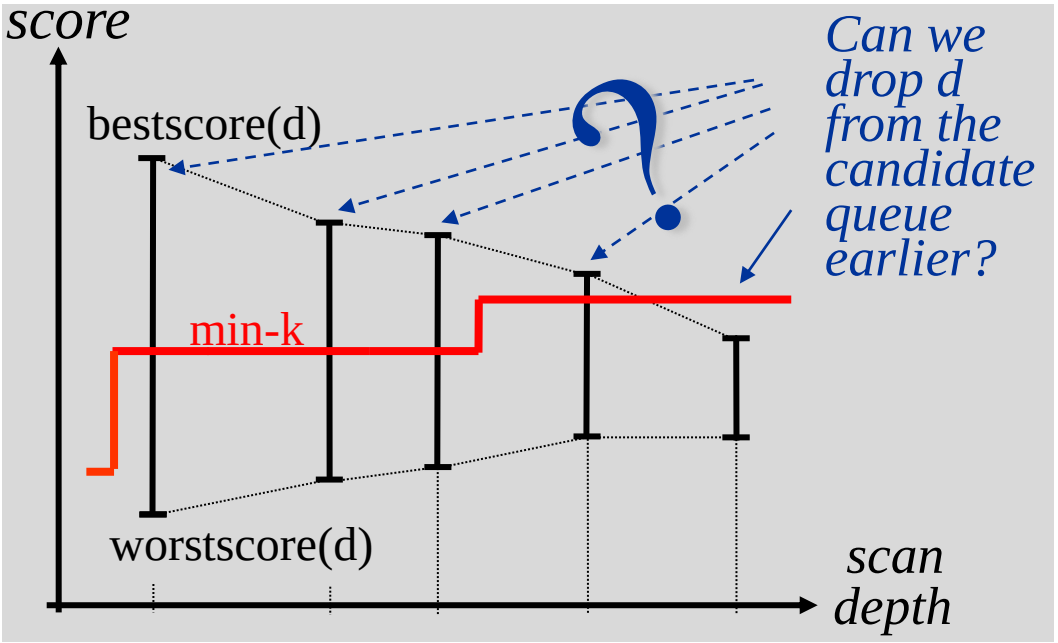
- Let E(d) denote the query dimensions at which d is *evaluated*, and let $high_i$ be the *high score* at the current scan position.
- TA family of algorithms is based on following invariant (using sum as aggr.):

$$\underbrace{\sum_{i \in E(d)} s_i(t_i,d)}_{t_i \in q} \leq score(d,q) \leq \underbrace{\sum_{i \in E(d)} s_i(t_i,d)}_{t_i \in q} + \underbrace{\sum_{i \notin E(d)} high_i}_{t_i \in q}$$

worstscore(d) | bestscore(d)

- Add d to top-k result, if worstscore(d) > min-k.
- Drop d only if bestscore(d) ≤ min-k, otherwise keep in candidate queue.

→ Often overly conservative in pruning, thus resulting in very long scans of the index lists (esp. for NRA)!



Probabilistic Threshold Test

[M. Theobald et al.: VLDB'04]

- Instead of using exact $high_i$ bounds, use RV's S_i to estimate the probability that d will exceed a certain score mass in its remaining lists:

$$p(d) := P\left[\left(\sum_{\substack{i \in E(d) \\ t_i \in q}} s_i(t_i, d) + \sum_{\substack{i \notin E(d) \\ t_i \in q}} S_i\right) > \delta\right]$$

with δ = min-k at current scan position

Probabilistic threshold test:

Drop candidate d from queue if $p(d) \leq \varepsilon$

Probabilistic guarantee: (for precision/recall@k)

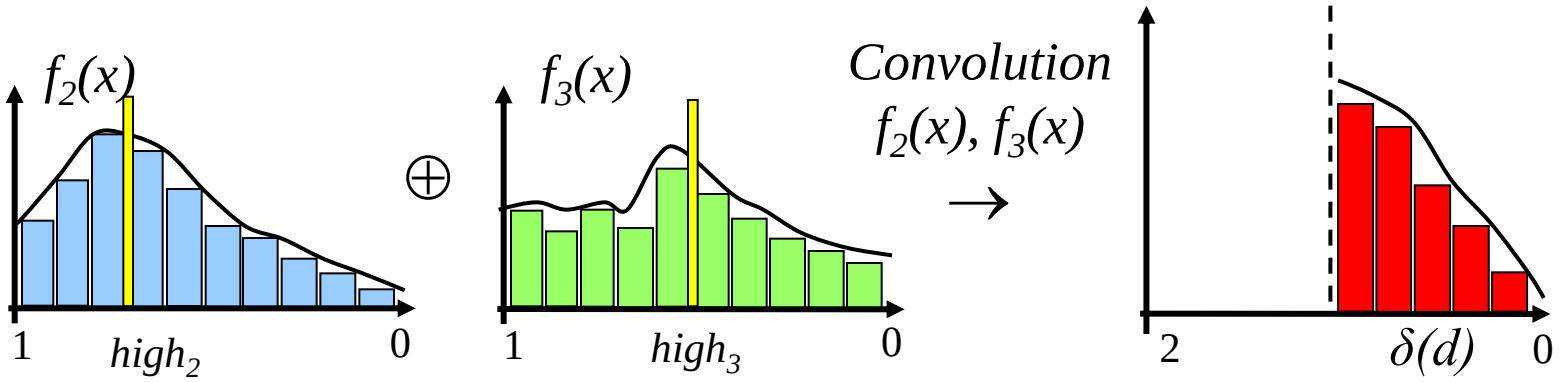
Binomial distribution of true (r) and false (k-r) top-k answers, using ε as upper bound for the probability of missing a true top-k result:

$$P[precision = \frac{r}{k}] = P[recall = \frac{r}{k}] = \binom{k}{r}(1 - p_{miss})^r \, p_{miss}^{(k-r)} \leq \binom{k}{r}(1-\varepsilon)^r \, \varepsilon^{(k-r)}$$

$\Rightarrow$ E[precision@k] = E[recall@k] = 1 – ε

Probabilistic Score Predicator

Candidate d with
$2 \notin E(d)$,
$3 \notin E(d)$



- Postulating **Uniform** or **Pareto** score distribution in [0, $high_i$]
  - Compute convolution using moment-generating function
  - Use Chernoff-Hoeffding tail bounds (see Chapter I.1) or generalized bounds for correlated dimensions (Siegel 1995)
- Fitting **Poisson** distribution or **Poisson mixture**
  - Over equidistant values: $P[d = v_j] = e^{-\alpha_i} \frac{\alpha_i^{j-1}}{(j-1)!}$
  - Easy and exact convolution!
- Score distribution approximated by **histograms**:
  - Precomputed for each dimension (e.g. equi-width with n cells 0..n-1)
  - Pair-wise convolution at query-execution time (producing 2n cells 0..2n-1)

$$H_{X+Y}[k].freq = \sum_{i=0}^{k} H_X[i].freq \cdot H_Y[k-i].freq$$

Flexible Scheduling of SA's and RA's

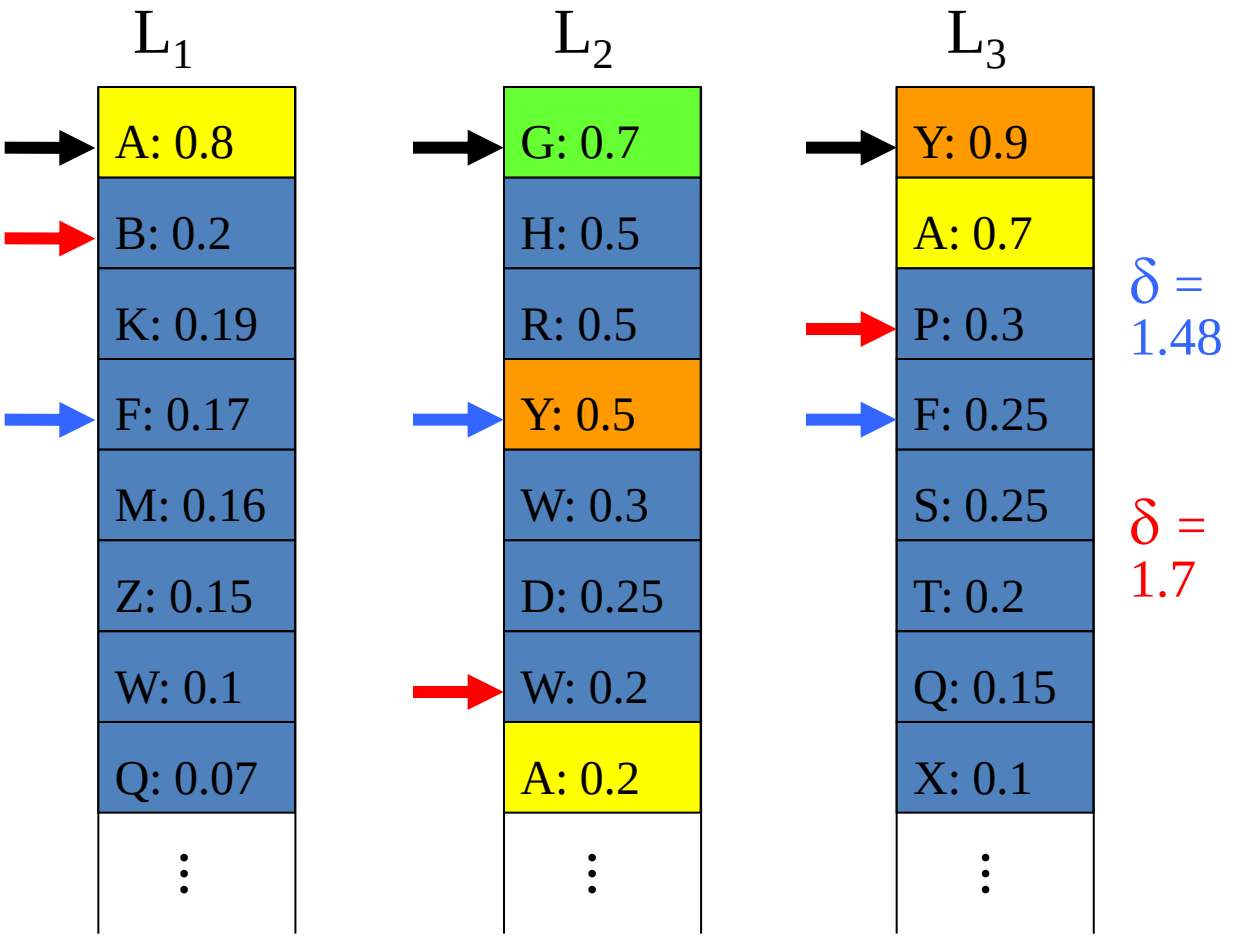
Goals:

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 (again, heuristics) 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

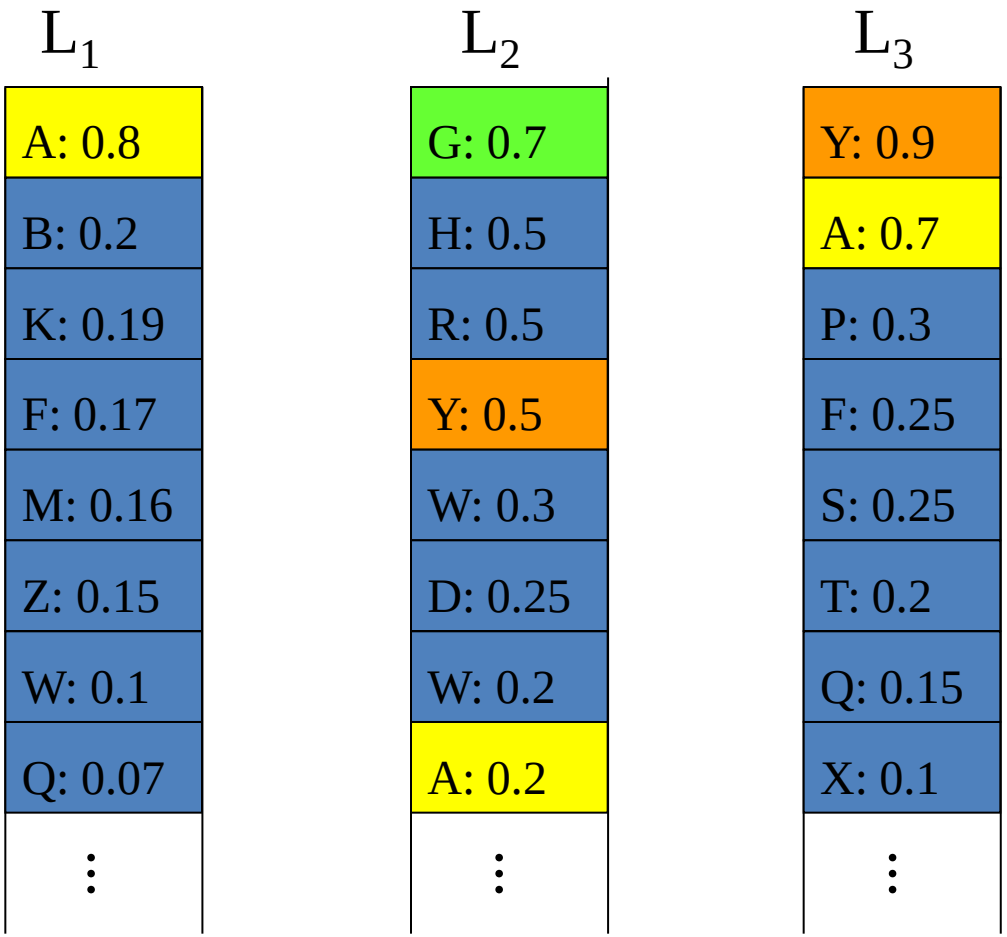
Advanced Scheduling Example



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

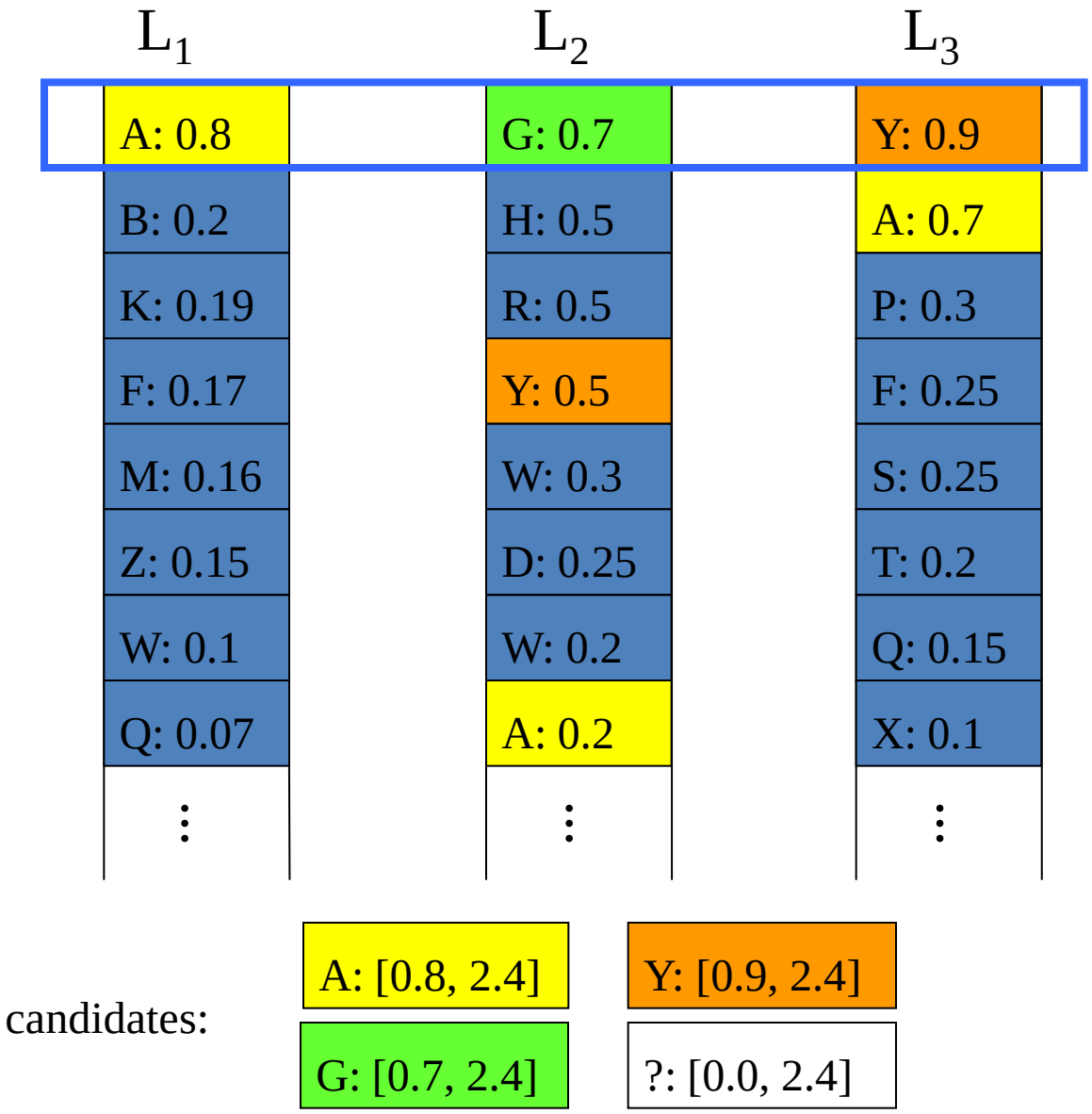
+ carefully chosen RAs:
score lookups for "interesting" candidates

Advanced Scheduling Example

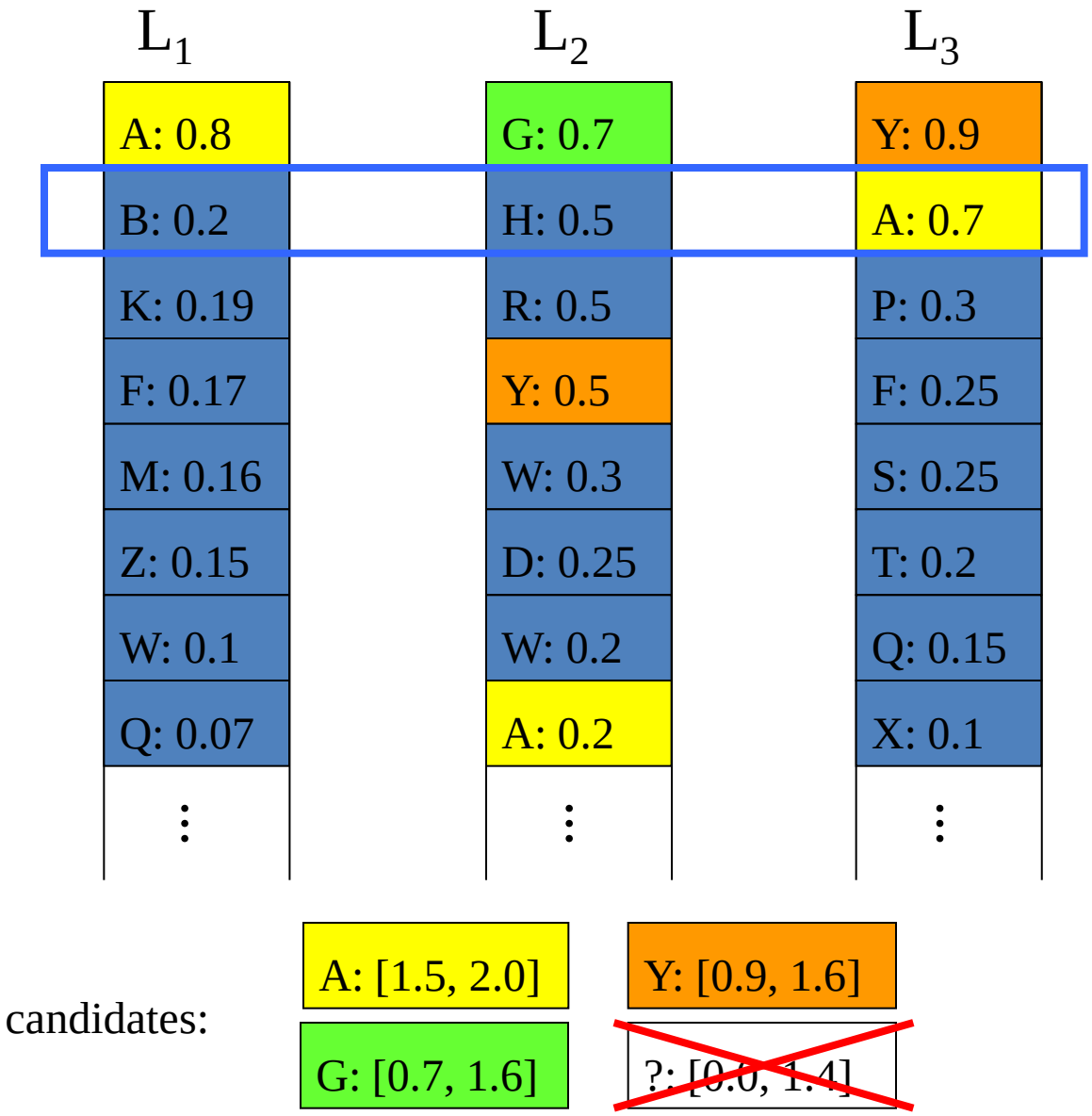


compute top-1 result
using flexible SAs and RAs

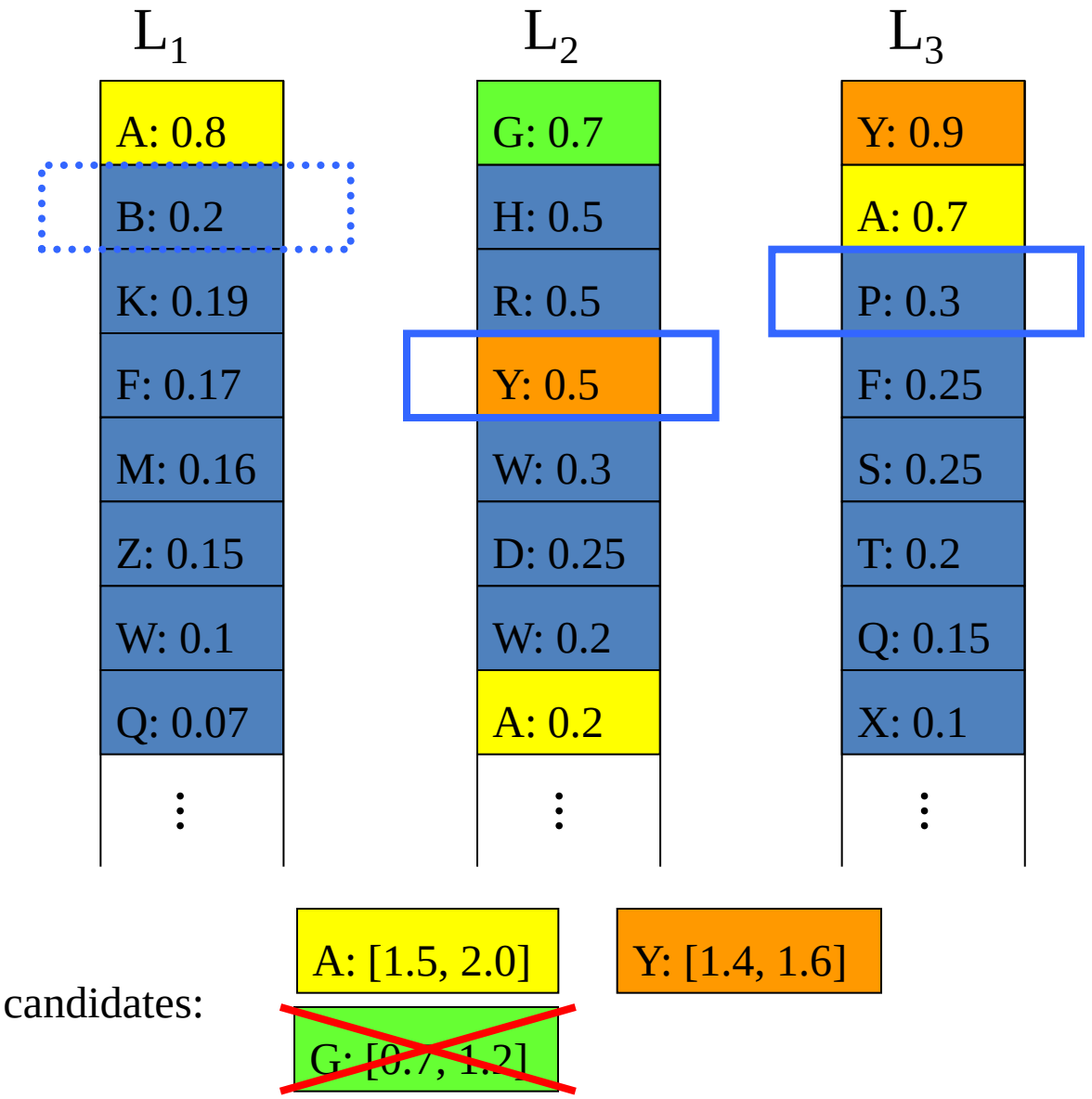
Advanced Scheduling Example



Advanced Scheduling Example



Advanced Scheduling Example



IO-Top-k: Index-Access Optimized Top-k

[Bast et al.: VLDB'06]

For **SA scheduling** plan next $b_1, ..., b_m$ index scan steps
for **batch of b steps** overall s.t. $\Sigma_{i=1..m} \; b_i = b$
and benefit($b_1, ..., b_m$) is max!

→ Solve **Knapsack-style** NP-hard problem for small batches of scans, or use greedy heuristics for larger batches.

Perform **additional RAs** when helpful:
1) To increase min-k (increase worstscore of d ∈ top-k), or
2) to prune candidates (decrease bestscore of d among candidates).

Last Probing (2-Phase Schedule):
Perform RAs for all candidates when
cost of remaining RAs ≤ (estimated) cost of remaining SAs
with score-prediction & cost model for deciding RA order.

Algorithm NRA

Fagin's NRA Algorithm:

List 1	List 2	List 3
doc 25 0.6	doc 17 0.6	doc 83 0.9
doc 78 0.5	doc 38 0.6	doc 17 0.7
doc 83 0.4	doc 14 0.6	doc 61 0.3
doc 17 0.3	doc 5 0.6	doc 81 0.2
doc 21 0.2	doc 83 0.5	doc 65 0.1
doc 91 0.1	doc 21 0.3	doc 10 0.1
	doc 44 0.1	

Algorithm NRA

read one doc from every list

round 1

List 1	List 2	List 3
doc 25 0.6	doc 17 0.6	doc 83 0.9
doc 78 0.5	doc 38 0.6	doc 17 0.7
doc 83 0.4	doc 14 0.6	doc 61 0.3
doc 17 0.3	doc 5 0.6	doc 81 0.2
doc 21 0.2	doc 83 0.5	doc 65 0.1
doc 91 0.1	doc 21 0.3	doc 10 0.1
	doc 44 0.1	

lists sorted by score

current score

best-score

Candidates
doc 83 [0.9, **2.1**]
doc 17 [0.6, **2.1**]
doc 25 [0.6, **2.1**]

min top-2 score: 0.6

maximum score for unseen docs: 2.1

min-top-2 < best-score of candidates

Algorithm NRA

read one doc from every list

round 2

List 1	List 2	List 3
doc 25 0.6	doc 17 0.6	doc 83 0.9
doc 78 0.5	doc 38 0.6	doc 17 0.7
doc 83 0.4	doc 14 0.6	doc 61 0.3
doc 17 0.3	doc 5 0.6	doc 81 0.2
doc 21 0.2	doc 83 0.5	doc 65 0.1
doc 91 0.1	doc 21 0.3	doc 10 0.1
	doc 44 0.1	

lists sorted by score

Candidates

doc 17 [1.3, **1.8**]
doc 83 [0.9, **2.0**]
doc 25 [0.6, **1.9**]
doc 38 [0.6, **1.8**]
doc 78 [0.5, **1.8**]

min top-2 score: 0.9

maximum score for unseen docs: 1.8

min-top-2 < best-score of candidates

Algorithm NRA

read one doc from every list

round 3

List 1	List 2	List 3
doc 25 0.6	doc 17 0.6	doc 83 0.9
doc 78 0.5	doc 38 0.6	doc 17 0.7
doc 83 0.4	doc 14 0.6	doc 61 0.3
doc 17 0.3	doc 5 0.6	doc 81 0.2
doc 21 0.2	doc 83 0.5	doc 65 0.1
doc 91 0.1	doc 21 0.3	doc 10 0.1
	doc 44 0.1	

Candidates

doc 83 [1.3, **1.9**]
doc 17 [1.3, **1.9**]
doc 25 [0.6, **1.5**]
doc 78 [0.5, **1.4**]

min top-2 score: 1.3

maximum score for unseen docs: 1.3

min-top-2 < best-score of candidates

no more new docs can get into top-2

but, extra candidates left in queue

Algorithm NRA

read one doc from every list

round 4

List 1	List 2	List 3
doc 25 0.6	doc 17 0.6	doc 83 0.9
doc 78 0.5	doc 38 0.6	doc 17 0.7
doc 83 0.4	doc 14 0.6	doc 61 0.3
doc 17 0.3	doc 5 0.6	doc 81 0.2
doc 21 0.2	doc 83 0.5	doc 65 0.1
doc 91 0.1	doc 21 0.3	doc 10 0.1
	doc 44 0.1	

lists sorted by score

Candidates

doc 17 **1.6**
doc 83 [1.3, **1.9**]
doc 25 [0.6, **1.4**]

min top-2 score: 1.3

maximum score for unseen docs: 1.1

min-top-2 < best-score of candidates

no more new docs can get into top-2

but, extra candidates left in queue

Algorithm NRA

read one doc from every list

round 5

List 1	List 2	List 3
doc 25 0.6	doc 17 0.6	doc 83 0.9
doc 78 0.5	doc 38 0.6	doc 17 0.7
doc 83 0.4	doc 14 0.6	doc 61 0.3
doc 17 0.3	doc 5 0.6	doc 81 0.2
doc 21 0.2	doc 83 0.5	doc 65 0.1
doc 91 0.1	doc 21 0.3	doc 10 0.1
	doc 44 0.1	

Candidates

doc 83 **1.8**
doc 17 **1.6**

min top-2 score: 1.6

maximum score for unseen docs: 0.8

no extra candidate in queue

Done!

Algorithm IO-Top-k

not necessarily round robin

List 1	List 2	List 3
doc 25 0.6	doc 17 0.6	doc 83 0.9
doc 78 0.5	doc 38 0.6	doc 17 0.7
doc 83 0.4	doc 14 0.6	doc 61 0.3
doc 17 0.3	doc 5 0.6	doc 81 0.2
doc 21 0.2	doc 83 0.5	doc 65 0.1
doc 91 0.1	doc 21 0.3	doc 10 0.1
	doc 44 0.1	

Round 1: same as NRA

Candidates

doc 83 [0.9, **2.1**]
doc 17 [0.6, **2.1**]
doc 25 [0.6, **2.1**]

min top-2 score: 0.6

maximum score for unseen docs: 2.1

min-top-2 < best-score of candidates

Algorithm IO-Top-k

not necessarily round robin

List 1	List 2	List 3
doc 25 0.6	doc 17 0.6	doc 83 0.9
doc 78 0.5	doc 38 0.6	doc 17 0.7
doc 83 0.4	doc 14 0.6	doc 61 0.3
doc 17 0.3	doc 5 0.6	doc 81 0.2
doc 21 0.2	doc 83 0.5	doc 65 0.1
doc 91 0.1	doc 21 0.3	doc 10 0.1
	doc 44 0.1	

Round 2

Candidates

doc 17 [1.3, **1.8**]
doc 83 [0.9, **2.0**]
doc 25 [0.6, **1.9**]
doc 78 [0.5, **1.4**]

min top-2 score: 0.9

maximum score for unseen docs: 1.4

min-top-2 < best-score of candidates

Algorithm IO-Top-k

not necessarily round robin

List 1	List 2	List 3
doc 25 0.6	doc 17 0.6	doc 83 0.9
doc 78 0.5	doc 38 0.6	doc 17 0.7
doc 83 0.4	doc 14 0.6	doc 61 0.3
doc 17 0.3	doc 5 0.6	doc 81 0.2
doc 21 0.2	doc 83 0.5	doc 65 0.1
doc 91 0.1	doc 21 0.3	doc 10 0.1
	doc 44 0.1	

lists sorted by score

Round 3

Candidates

doc 17 1.6
doc 83 [1.3, **1.9**]
doc 25 [0.6, **1.4**]

min top-2 score: 1.3

maximum score for unseen docs: 1.1

min-top-2 < best-score of candidates

potential candidate for top-2

Algorithm IO-Top-k

not necessarily round robin

List 1	List 2	List 3
doc 25 0.6	doc 17 0.6	doc 83 0.9
doc 78 0.5	doc 38 0.6	doc 17 0.7
doc 83 0.4	doc 14 0.6	doc 61 0.3
doc 17 0.3	doc 5 0.6	doc 81 0.2
doc 21 0.2	doc 83 0.5	[illegible] 0.1
doc 91 0.1	doc 21 0.3	doc 10 0.1
	doc 44 0.1	

lists sorted by score

random access for doc 83

Round 4: random access for doc 83

Candidates

doc 83 **1.8**
doc 17 **1.6**

min top-2 score: 1.6

maximum score for unseen docs: 1.1

no extra candidate in queue

Done!

→ fewer sorted accesses

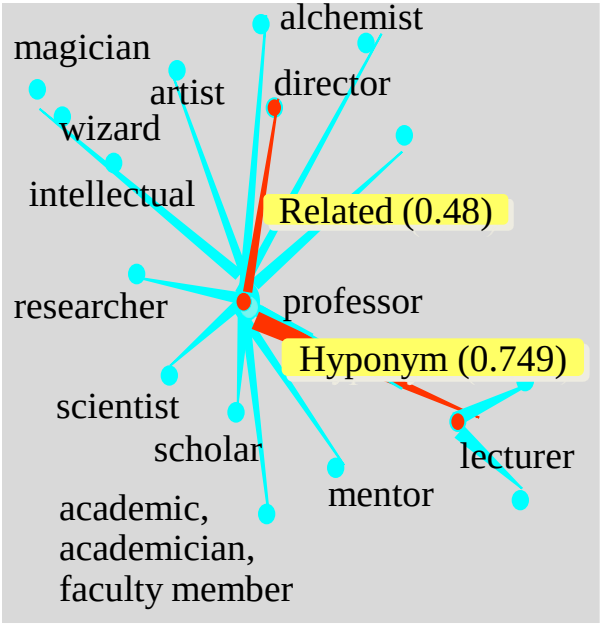
→ carefully scheduled random access

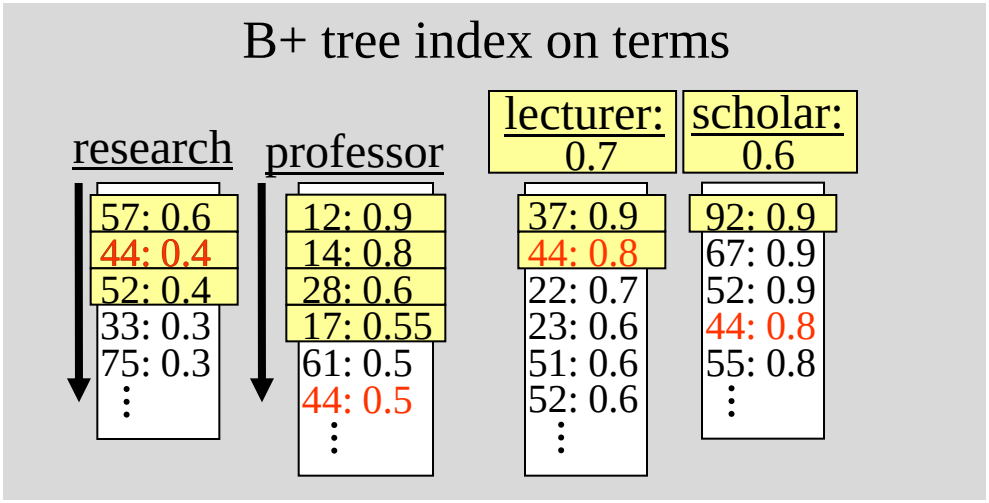
Query Expansion with Incremental Merging

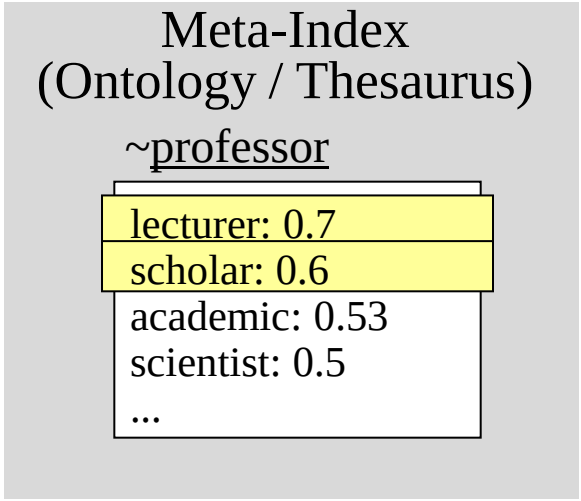
[Theobald et al.: SIGIR'05]

Expanded query **q: ~"*professor*" *research*** with expansions $exp(t_i)=\{ci_j \mid sim(t_i,c_{ij}) \geq \theta, t_i \in q\}$ based on **ontological similarity** modulating monotonic score aggregation, e.g., using **weighted sum** over $sim(t_i,c_{ij})*s(c_{ij},d_l)$.

Incrementally merge index lists instead of computing full expansion!







→ efficient, robust, self-tuning

Query Expansion with Incremental Merging

[Theobald et al.: SIGIR 2005]

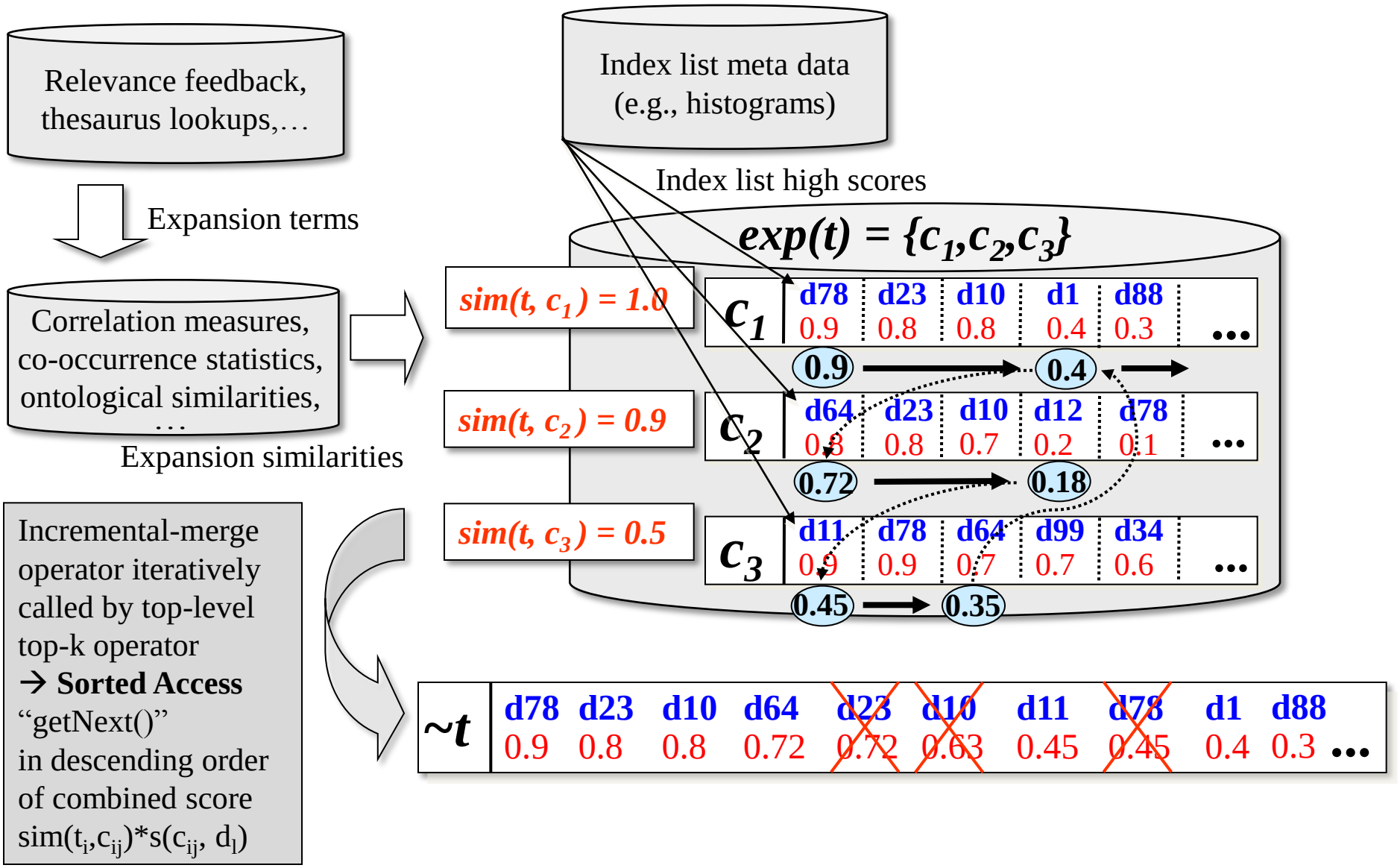
Principles of the algorithm:

- Organize **possible expansions** $exp(t)=\{c_j \mid sim(t,c_j) \geq \theta, t \in q\}$ in priority queues based on $sim(t,c_j)$ for each t.

For a given $q = \{t_1, t_m\}$ do:

- Keep track of **active expansions** for each t_i:
  $actExp(t_i) = \{c_{i1}, c_{i2}, ...c_{ij}\}$
- Scan index lists L_{ij} maintaining cursors $pos(L_{ij})$ and score bounds $high(L_{ij})$ for each list.
- At each scan step and for each t_i do:
  Advance cursor of L_{ij} for $c_{ij} \in actExp(t_i)$ for which
  $best := high(L_{ij})*sim(t_i,c_{ij})$ is largest among all of $exp(t_i)$
  or add next largest $c_{ij'}$ to $actExp(t_i)$ if $high(L_{ij'})*sim(t_i,c_{ij'}) > best$
  and proceed in this new list.

Incremental-Merge Operator



Top-k Queries on Internet Sources

[Marian et al.: TODS'04]

Setting:

- Score-ordered lists **dynamically produced** by Internet sources.
- Some sources restricted to RA lookups only (no sorted lists).

Example:

Preference search for hotel based on *distance*, *price*, *rating*
using *mapquest.com*, *hotelbooking.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 (basic 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 Limit k
```

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
...			

Open Source Search Engines

Lucene (http://lucene.apache.org/)

- The currently most commonly used open-source engine for IR
- Java implementation, easy to extend, e.g., by custom ranking functions
- Supports multiple document fields and (flat) XML documents → SLOR extension
- Used officially by Wikipedia

Indri (http://www.lemurproject.org/)

- Academic IR system by CMU & U Mass
- C++ implementation
- Built-in support for many common IR extensions such as relevance feedback

Wumpus (http://www.wumpus-search.org/)

- Academic IR system by University of Waterloo
- No a-priori documents but arbitrary parts of the corpus as retrieval units
- Supports probabilistic IR (BM25), language models, etc. for ranking

PF/Tijah (http://dbappl.cs.utwente.nl/pftijah/)

- XML-IR system developed at U Twente
- Very efficient column-store DBMS (MonetDB, native C) as storage backend

Additional Literature for Chapters V.3

Fast top-k search:

- C. Buckley, A. F. Lewit: Optimization of Inverted Vector Searches. SIGIR 1985
- A. Moffat, J. Zobel: Self-Indexing Inverted Files for Fast Text Retrieval, TOIS 1996
- X. Long, T. Suel: Optimized Query Execution in Large Search Engines with Global Page Ordering, VLDB 2003
- U. Güntzer, W.-T. Balke, Werner Kießling: Optimizing Multi-Feature Queries for Image Databases. VLDB 2000
- U. Güntzer, W.-T. Balke, W. Kießling: Towards Efficient Multi-Feature Queries in Heterogeneous Environments. ITCC 2001
- R. Fagin, A. Lotem, M. Naor: Optimal Aggregation Algorithms for Middleware, Journal of Computer and System Sciences 2003
- 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
- Marian, N. Bruno, L. Gravano: Evaluating Top-k Queries over Web-accessible Databases. TODS 2004
- M. Theobald, G. Weikum, R. Schenkel: Top-k Query Processing with Probabilistic Guarantees, VLDB 2004
- Martin Theobald, Ralf Schenkel, Gerhard Weikum: Efficient and self-tuning incremental query expansion for top-k query processing. SIGIR 2005
- H. Bast, D. Majumdar, R. Schenkel, M. Theobald, G. Weikum: IO-Top-k: Index-access Optimized Top-k Query Processing. VLDB 2006

V.4 Efficient Similarity Search

Given a document d: find all similar documents d'

(i.e., either exact or near duplicates)

with sim(d, d') $\geq \tau$

- Construct **representation** of d

  Set/bag of terms or N-grams (called "Shingles" when using N-grams at word level), set of links, link anchors, set of query terms that led to clicking d, etc.

- Define **similarity measure**

  Set overlap, Dice coeff., Jaccard coeff., Cosine, Hamming, etc.

- Deduplication **algorithm**

  For corpus of n documents: efficiently estimate pair-wise similarities over carefully designed index structures.

  → Sorting- and indexing-based approaches: $O(n^2)$ runtime

  → Similarity hashing (Min-Hashing & LSH): O(n) runtime

Example: Near-Duplicate News Articles

SFGate.com

SFGATE HOME · NEWS NEW! · BUSINESS · SPORTS · ENTERTAINMENT · TRAVEL

CLASSIFIEDS · JOBS · REAL ESTATE · CARS

SEARCH SFGate Web Search by YAHOO! Sign In | Register

Associated Press

Obama Takes on Question of Faith

By NEDRA PICKLER, Associated Press Writer
Monday, January 21, 2008

PRINTABLE E-MAIL SHARE COMMENTS (0) FONT | SIZE: TOOLS SPONSOR: verizon wireless

(01-21) 04:22 PST Columbia, S.C. (AP) --

Barack **Obama** is stepping up his effort to correct the misconception that he's a Muslim now that the presidential campaign has hit the Bible Belt.

At a rally to kick off a weeklong campaign for the South Carolina primary, **Obama** tried to set the record straight from an attack circulating widely on the Internet that is designed to play into prejudices against Muslims and fears of terrorism.

MOST READ | MOST E-MAILED | TOP STORIES

1. TGI Friday's employee found slain in San Mateo restaurant
2. Girl shot to death in Oakland by boy trying to scare her, police say
3. 5 Dead As Planes Collide in SoCal
4. Rainy week ahead for Bay Area, with snow on the hills
5. Cranky Pants traded in for Gary Coleman's
6. Giants want to develop lot next to AT&T Park
7. More men turning to implants for chests of gold

HOME PAGE | MY TIMES | TODAY'S PAPER | VIDEO | MOST POPULAR | TIMES TOPICS

Get Home Delivery | Log In | Register Now

The New York Times U.S.

U.S. All NYT Search

Ameriprise Financial

WORLD | U.S. | N.Y. / REGION | BUSINESS | TECHNOLOGY | SCIENCE | HEALTH | SPORTS | OPINION | ARTS | STYLE | TRAVEL | JOBS | REAL ESTATE | AUTOS

POLITICS | WASHINGTON | EDUCATION

Obama Takes on Question of Faith

By THE ASSOCIATED PRESS
Published: January 21, 2008

Filed at 7:16 a.m. ET

COLUMBIA, S.C. (AP) -- Barack Obama is stepping up his effort to correct the misconception that he's a Muslim now that the presidential campaign has hit the Bible Belt.

At a rally to kick off a weeklong campaign for the South Carolina primary, Obama tried to set the record straight from an attack circulating widely on the Internet that is designed to play into prejudices against Muslims and fears of terrorism.

SIGN IN TO E-MAIL OR SAVE THIS

PRINT

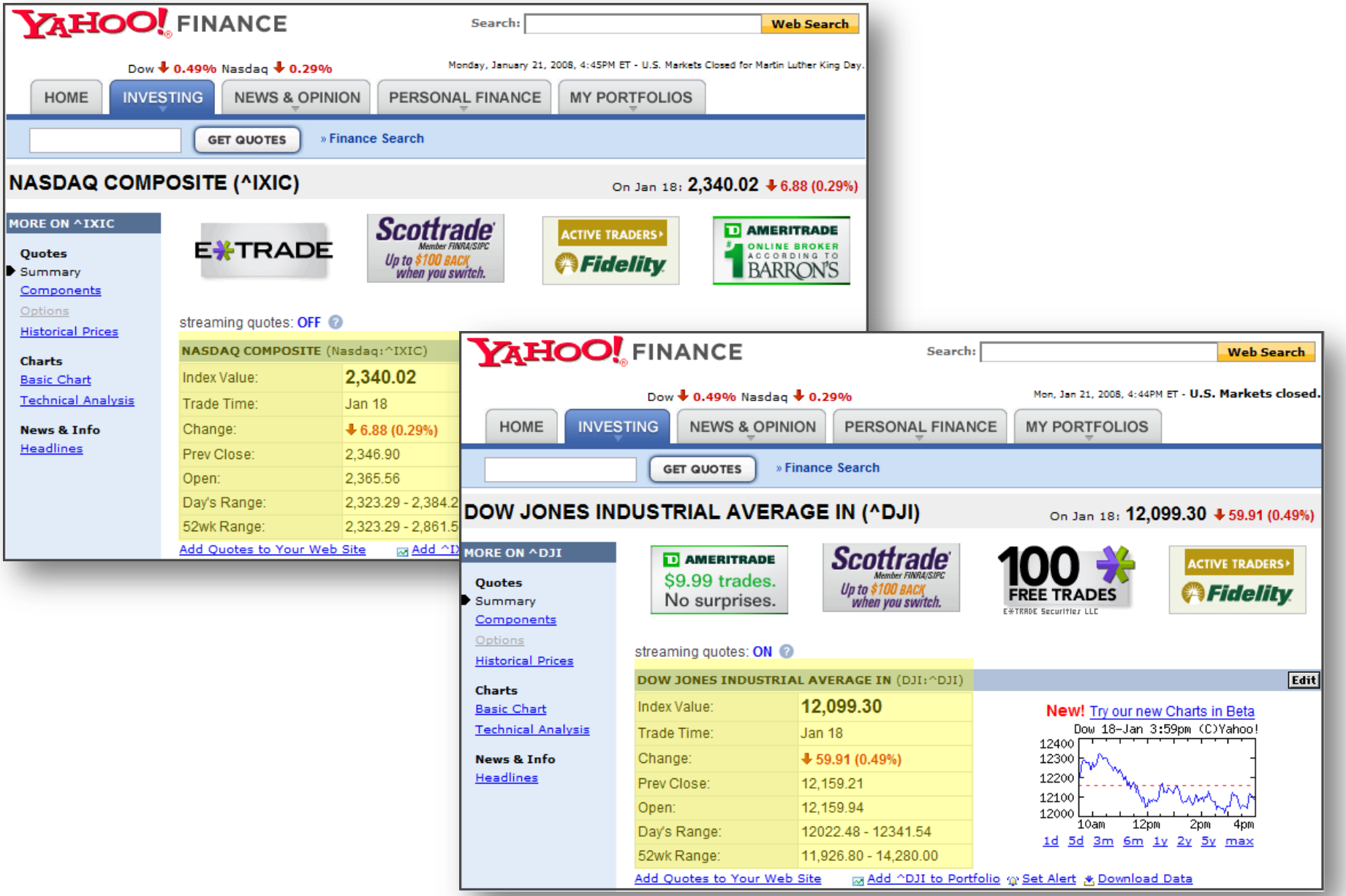
ARTICLE TOOLS SPONSORED BY THE SAVAGES

MOST POPULAR

E-MAILED | BLOGGED | SEARCHED

1. Nicholas D. Kristof: Hillary, Barack, Experience
2. Paul Krugman: Debunking the Reagan Myth
3. Pregnancy Problems Tied to Caffeine
4. Maureen Dowd: Red, White and Blue Tag Sale
5. Roger Cohen: U.S. Soldiers and Shoppers Hit the Wall
6. Stocks Plunge Worldwide on Fears of a U.S. Recession
7. New York Measuring Teachers by Test Scores
8. Op-Ed Contributor: Radical Love Gets a Holiday
9. A Cutting Tradition

Example: Near-Duplicate Stock Articles



SpotSigs Algorithm: *Divide & Conquer*

[Theobald et al.: SIGIR'08]

1) Pick a specific similarity measure: Jaccard

$$sim(A, B) = |A \cap B| / |A \cup B|$$

2) Develop an upper bound for the similarity of two documents based on a simple property:

   → Check difference in cardinality of signature sets |A|, |B|

3) Partition the collection into near-duplicate candidates based on this bound.

4) For each document: process only the relevant partitions and prune partitions as early as possible.

Upper bound for Jaccard Similarity

- Consider Jaccard similarity

$$sim(A, B) = |A \cap B| / |A \cup B|$$

- Generic upper bound for Jaccard:

$$sim(A, B) = \frac{|A \cap B|}{|A \cup B|} \leq \frac{\min(|A|, |B|)}{\max(|A|, |B|)}$$

$$|A \cap B| \leq \min(|A|, |B|)$$

$$|A \cup B| \geq \max(|A|, |B|)$$

$$sim(A, B) \leq \frac{|A|}{|B|}$$

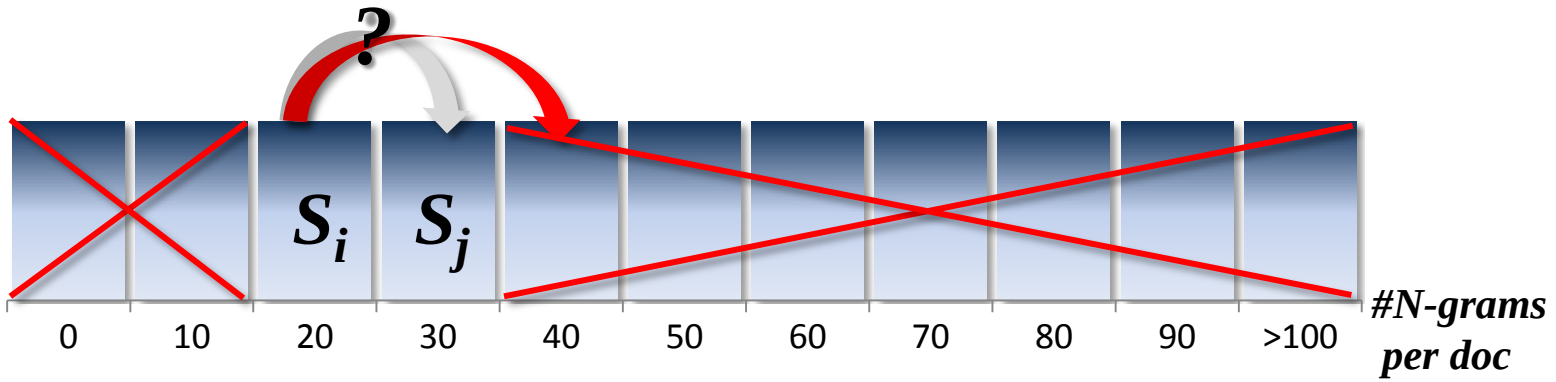
(assuming $|B| \geq |A|$, w.l.o.g.)

→ Do not compare any two signature sets A, B with $|A|/|B| < \tau$ or equivalently: $|B| - |A| > (1-\tau)\ |B|$

Partitioning the Collection

Given a similarity threshold τ, partition the documents (based on their signature set cardinality) into partitions $S_1,\ldots S_k$, such that:

1) Any potentially similar pair is within the same or at most two subsequent partitions (→ no false negatives), and
2) no non-similar pair is within the same partition (→ no false positives w.r.t. signature cardinality).



SpotSigs Algorithm:

Index each partition S_i using an inverted index over N-grams.

Deduplicate entire set of potential matches in one scan of the inverted index for S_i and neighboring index for S_j.

Min-Wise Independent Permutations (MIPs)

[Gionis, Indyk, Motwani: VLDB'99]

Use approximations based on overlapping N-grams (e.g., shingles) and statistical estimators:

→ **Min-wise independent permutations / Min-Hash method**:
Compute min(π(D)), min(π(D')) for random permutations π of N-gram sets D and D' of docs d and d' and check whether min(π(D)) = min(π(D')).

For example using 2-grams:

3: Barack$Obama
8: Obama$is
12: is$stepping
17: stepping$up …

Input set/vector of N-gram ids
(typically sparse)

D:	**3**	**8**	**12**	**17**	**21**	**24**

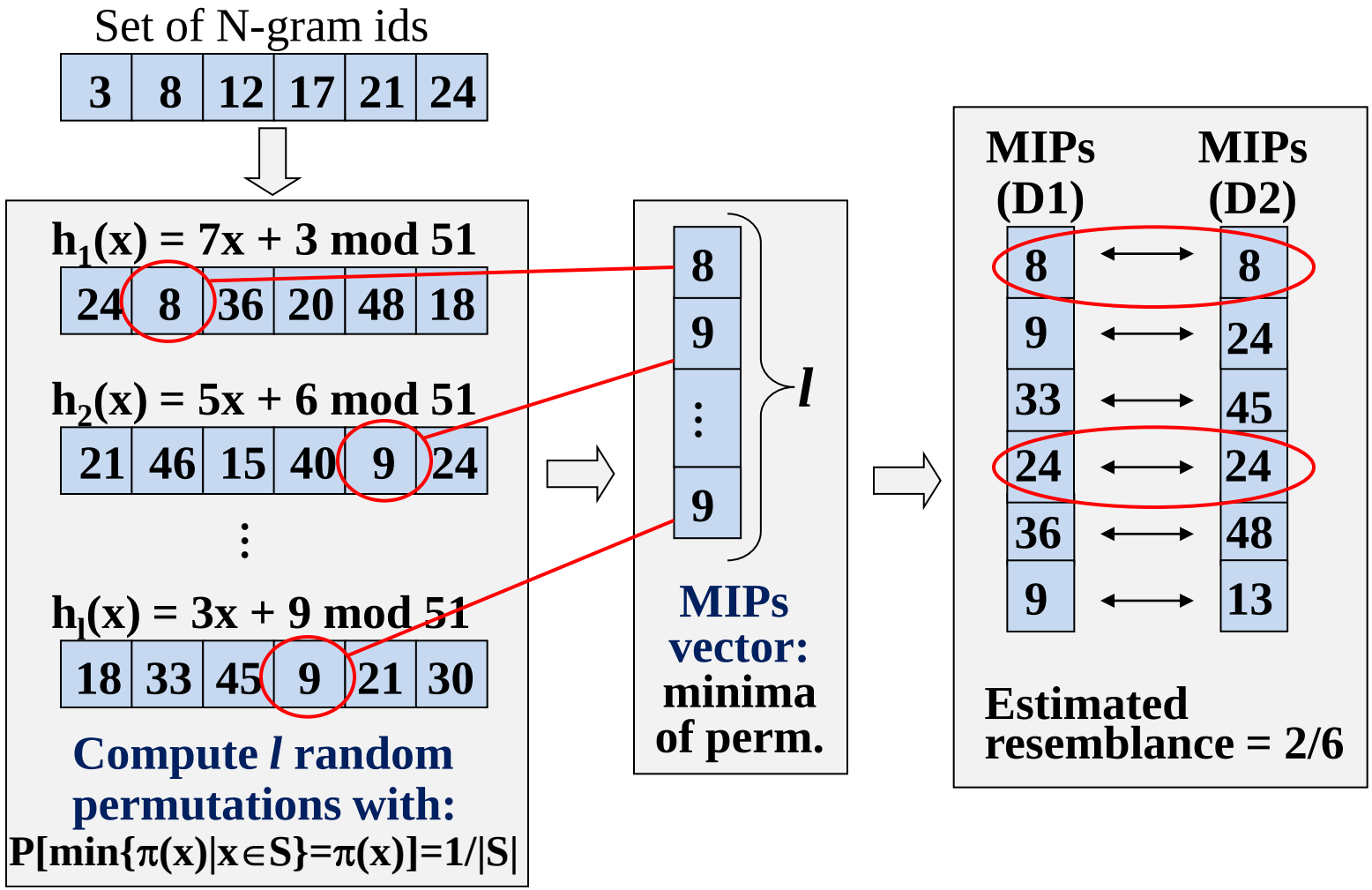
π_1: $h_1(x) = 7x + 3 \bmod 51$

Random permutation
(e.g., linear transformation of input dimensions)

π_1 (D):	**24**	**8**	**36**	**20**	**48**	**18**

Dimensionality of output vector

Min-Hashing Example



MIPs are an **unbiased estimator of set resemblance**:

$$P\,[\min\{h(x) \mid x \in A\} = \min\{h(y) \mid y \in B\}] = |A \cap B| / |A \cup B|$$

MIPs can be viewed as repeated random sampling of x, y from A, B.

Near-Duplicate Elimination

[Broder et al. 1997]

Duplicates on the Web may be slightly perturbed.
Crawler & indexing interested in identifying **near-duplicates** to **reduce redundancy** among search results.

Approach:

- Represent each document d as set (or sequence) of Shingles (N-grams over tokens).
- Encode Shingles by hash fingerprints (e.g., using SHA-1), yielding set of numbers $S(d) \subseteq [1..n]$ with, e.g., $n=2^{64}$.
- Compare two docs d, d' that are suspected to be duplicates by:
  - **Resemblance:** $\frac{|S(d) \cap S(d')|}{|S(d) \cup S(d')|}$ Jaccard coefficient
  - **Containment:** $\frac{|S(d) \cap S(d')|}{|S(d)|}$ Relative overlap
- Drop d' from search index if resemblance or containment is above threshold.

Efficient Duplicate Detection in Large Corpora

[Broder et al. 1997]

Avoid comparing *all* pairs of documents:

Solution: Shingle-based Clustering

1) For each doc compute shingle-set (and MIPs)
2) Produce (shingleId, docId) sorted list.
3) Produce (docId1, docId2, shingleCount) table with counters for common shingles.
4) Identify (docId1, docId2) pairs with shingleCount above threshold and add (docId1, docId2) edge to graph.
5) Compute connected components of graph (union-find) → these are the near-duplicate clusters!

Trick for additional speedup of steps 2 and 3:

- Compute super-shingles (meta sketches) for shingles of each doc.
- Docs with many common shingles have common super-shingle w.h.p.

Locality Sensitive Hashing (with Min-Hashing)

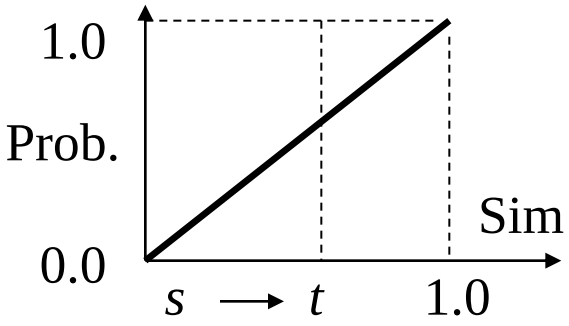
- Key idea: Hash each document multiple times!
  - Choose k independent Min-Hash permutations π_{ij} and concatenate $\pi_{i1}(D) \oplus \pi_{i2}(D) \oplus \ldots \pi_{ik}(D)$ into a longer signature $C_i(D)$ (→ good for precision!)
  - Employ l randomly chosen (but fixed) concatenations $C_1(D),\ldots,C_l(D)$ for detecting collisions among similar documents (→ good for recall!)

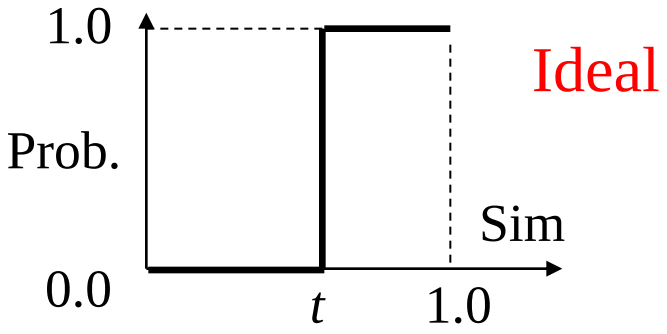
→ Two documents are near duplicates if they are hashed into the same bucket by any of the C_i, i.e., $C_i(D) = C_j(D')$ for any $i,j < l$.

LSH Properties

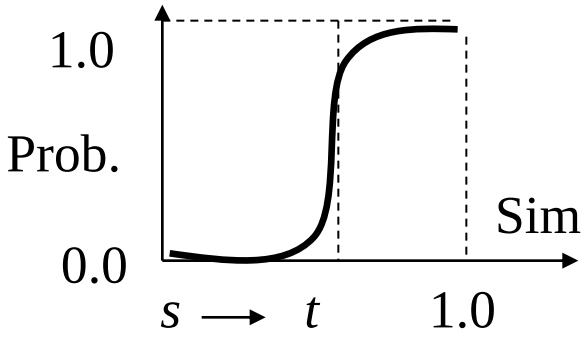
- Example: Find pairs of documents with similarity $s \geq t$
- Suppose we use only one Min-Hash function consisting of a single random permutation π.

  (Recall that the probability of a collision of two documents corresponds to the Jaccard similarity of their signature sets.)





- However, partitioning into *l* concatenations of k Min-Hashes yields:



- That is, for a similarity of 80%, the probability of missing the pair is $(1\text{-}0.8^k)^l = 3.5\text{x}10^{-4}$ for k=5 and l=20.

Additional Literature for Chapter V.4

Similarity search:

- A. Broder, S. Glassman, M. Manasse, G. Zweig: Syntactic Clustering of the Web, WWW 1997
- A. Gionis, P. Indyk, R. Motwani: Similarity Search in High Dimensions via Hashing. VLDB 1999
- M. Henzinger: Finding Near-Duplicate Web Pages: a Large-Scale Evaluation of Algorithms, SIGIR 2006
- M. Theobald, J. Siddharth, A. Paepcke: SpotSigs: Robust and efficient near duplicate detection in large web collections. SIGIR 2008

Summary of Chapter V

- Indexing by **inverted lists**:
  - docId-order or score-order with very effective compression
  - Per-term or per-doc partitioned across server farm for scalability
- Query processing by **index list merging** or **threshold algorithms** (TA, NRA, CA, etc.):
  → Large variety of top-k algorithms
  → Optionally with approximation, smart scheduling, etc.
- Efficient **similarity search** either by sorting & partitioning (*exact matching*) or by Min-Hashing & LSH (*approximate matching* but tunable to achieve excellent precision and recall values).