Indexing and Querying

Avishek Anand

credits: Some of the slides have been taken/inspired from Prof. Gerhard Weikum’s lecture IRDM 2009 (Saarland Uni) and Chris Manning (Stanford CS276)
Query Processing over Inverted Indexes

* Query Processing over Document-id ordered lists
  * Document-at-a-Time vs Term-at-a-Time Processing
  * WAND Processing

* Top-K Processing
  * Fagin’s top-k
  * TA, NRA and CA

* Supporting advanced queries - phrases, proximity-aware, temporal queries
Boolean Queries — keywords form a boolean expression

- Conjunctive, Disjunctive, Negation, combination
- (“spears” AND “britney”) OR (“lady” AND “gaga”)

Phrase queries — entire phrase present in the same order

- “to be or not be that is the question”

Advanced Queries

- Temporal queries — “summer olympics @ [2001 - 2003]”
- Proximity aware queries — keywords should be present within a gap of each other
- Wild card queries — “a * saved is a * earned”
Document Ordering

- Based on faster intersections
- High compression of index using gap encoding of dids
- Easily updatable

Score/Impact Ordering

- Based on processing Top-k results fast
- Low compression ratio
- Difficult to update

Index organisation depends on query processing style.
Best choice for boolean queries are document ordered lists

Postings are stored in a column major format

A list of all document identifiers — good for compression and faster boolean operations

A list of all scores

Scores are typically aggregated by weighted sum of the partial term scores
## Term-At-A-Time

| hannover | 12 | 23 | 48 | 71 | 93 | 96 | 101 |
| messe   | 18 | 23 | 71 | 77 | 112| 189|

- **Term-at-a-time**
  - Process one list at a time
  - Maintain accumulators for partial results and update them
  - Best for unions

**Accumulators in memory**
Term-At-A-Time

- Process one list at a time
- Maintain accumulators for partial results and update them
- Best for unions

![Accumulators in memory](image)
Term-At-A-Time

- Process one list at a time
- Maintain accumulators for partial results and update them
- Best for unions

hannover

| 12 | 23 | 48 | 71 | 93 | 96 | 101 |

messe

| 18 | 23 | 71 | 77 | 112 | 189 |

Accumulators in memory

| 18 | 12 | 71 |
| 23 | 77 | 93 |
| 48 | 189 | 101 | 112 |
| 96 |
**Document-At-A-Time**

- Open cursors to all lists
- Systematically move cursors to satisfy boolean expression
- Best for intersections

**Conjunctive query semantics**

- In each iteration find the max did M
- Move other cursors to greater or equal to M
- If all cursors point to M, move all one step further
**Document-At-A-Time**

- Open cursors to all lists
- Systematically move cursors to satisfy boolean expression
- Best for intersections

**Conjunctive query semantics**

- In each iteration find the max did M
- Move other cursors to greater or equal to M
- If all cursors point to M, move all one step further
Document-At-A-Time

- Open cursors to all lists
- Systematically move cursors to satisfy boolean expression
- Best for intersections

Conjunctive query semantics
- In each iteration find the max did M
- Move other cursors to greater or equal to M
- If all cursors point to M, move all one step further

hannover

| 12 | 23 | 48 | 71 | 93 | 96 | 101 |

messe

| 18 | 23 | 71 | 77 | 112 | 189 |
Skip Lists

| hannover | 12 | 23 | 48 | 71 | 93 | 96 | 101 | ...
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>messe</td>
<td>18</td>
<td>23</td>
<td>71</td>
<td>77</td>
<td>112</td>
<td>189</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

What if we are allowed to store extra information for each lists?
Skip Lists

What if we are allowed to store extra information for each lists?

- Skip list allow fast intersections acting as a secondary index over posting lists
- Typically skip over fixed number of postings and are square root of the length of the postings list
Skip Lists

What if we are allowed to store extra information for each list?

- Skip list allow fast intersections acting as a secondary index over posting lists.
- Typically skip over fixed number of postings and are square root of the length of the postings list.
Simple Computational Model

Average length of a posting list = n
Average number of terms in the query = m

1. What is the complexity (space and time) of both methods?

2. What method would you use for phrase queries e.g. “intel inside”?

3. How would you use the positional information for phrase queries?
Weak AND

- Generalized version of AND operation

\[
\sum_{1 \leq i \leq k} x_i w_i \geq \theta,
\]

for a query of \( k \) words

- Weights and indicator variables specify the degree of AND
- Threshold helps avoid computation of non important documents
- An instance of DAAT scoring
Assume a special iterator on the postings of the form “go to the first docID greater than X”
Assume a special iterator on the postings of the form “go to the first docID greater than X”

Typical state: we have a “cursor” at some docID in the postings of each query term
- Each cursor moves only to the right, to larger docIDs
Assume a special iterator on the postings of the form “go to the first docID greater than $X$”

Typical state: we have a “cursor” at some docID in the postings of each query term
  - Each cursor moves only to the right, to larger docIDs

Invariant – all docIDs lower than any cursor have already been processed, meaning
  - These docIDs are either pruned away or
  - Their scores have been computed
At all times for each query term $t$, we maintain an upper bound $UB_t$ on the score contribution of any doc to the right of the cursor.

- Max (over docs remaining in $t$’s postings) of $w_t(doc)$
- As cursor moves right, the UB drops

```
hannover  12  23  48  71  93  96  101
```

$UB(“hannover”) = 2.8$
Query: *catcher in the rye*

Let's say the current cursor positions are as below:

- **catcher**: 273
- **rye**: 304
- **in**: 589
- **the**: 762
Query: *catcher in the rye*

Let’s say the current cursor positions are as below:

- **catcher**: $UB_{catcher} = 2.3$
- **rye**: $UB_{rye} = 1.8$
- **in**: $UB_{in} = 3.3$
- **the**: $UB_{the} = 4.3$

Threshold = 6.8
Query: *catcher in the rye*

Let’s say the current cursor positions are as below

- **catcher**: UB\(_{catcher}\) = 2.3
- **rye**: UB\(_{rye}\) = 1.8
- **in**: UB\(_{in}\) = 3.3
- **the**: UB\(_{the}\) = 4.3

Threshold = 6.8
Query: *catcher in the rye*

Let’s say the current cursor positions are as below

- **catcher**: 273
- **rye**: 304
- **in**: 589
- **the**: 762

Threshold = 6.8

- $UB_{\text{catcher}} = 2.3$
- $UB_{\text{rye}} = 1.8$
- $UB_{\text{in}} = 3.3$
- $UB_{\text{the}} = 4.3$
Avishek Anand

mWAND Algorithm

- Terms sorted in order of cursor positions
- Move cursor to 589 or right

Threshold = 6.8

\[
\begin{align*}
\text{catcher} & : UB_{\text{catcher}} = 2.3 \\
\text{rye} & : UB_{\text{rye}} = 1.8 \\
\text{in} & : UB_{\text{in}} = 3.3 \\
\text{the} & : UB_{\text{the}} = 4.3
\end{align*}
\]
mWAND Algorithm

- Terms sorted in order of cursor positions
- Move cursor to 589 or right

**catcher**
- UB\text{catcher} = 2.3

**rye**
- UB\text{rye} = 1.8

**in**
- UB\text{in} = 3.3

**the**
- UB\text{the} = 4.3

Threshold = 6.8
mWAND Algorithm

- Terms sorted in order of cursor positions
- Move cursor to 589 or right

Threshold = 6.8

\[ UB_{\text{catcher}} = 2.3 \]
\[ UB_{\text{rye}} = 1.8 \]
\[ UB_{\text{in}} = 3.3 \]
\[ UB_{\text{the}} = 4.3 \]
- Terms sorted in order of cursor positions
- Move cursor to 589 or right

**catcher**
- 273 Hopeless docs

**rye**
- 304 Hopeless docs

**in**
- 589

**the**
- 762

Threshold = 6.8

- $UB_{catcher} = 2.3$
- $UB_{rye} = 1.8$
- $UB_{in} = 3.3$
- $UB_{the} = 4.3$

Update UB’s
• If 589 is present in enough postings, compute its full cosine score – else some cursors to right of 589

• Pivot again …
What happens if we advance cursors in only one posting list in each round?

In this scenario the cursor for the term “catcher” is advanced, and then a new pivot is found
Top-k Query Processing

- Result size are usually large and results are presented in descending score order

- Typically the top ranked results are interesting

- If the goal is to obtain only top-k results, we use top-k query processing techniques over score-ordered lists

- Postings are materialised in score order and row-major order

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>((d_1, 0.9))</td>
<td></td>
</tr>
<tr>
<td>((d_4, 0.5))</td>
<td></td>
</tr>
<tr>
<td>((d_{34}, 0.3))</td>
<td></td>
</tr>
<tr>
<td>....</td>
<td></td>
</tr>
</tbody>
</table>
Top-k Query Processing

- Result size are usually large and results are presented in descending score order

- Typically the top ranked results are interesting

- If the goal is to obtain only top-k results, we use top-k query processing techniques over score-ordered lists

- Postings are materialised in score order and row-major order

| (d_1, 0.9) |
| (d_4, 0.5) |
| (d_{34}, 0.3) |
| ....... |
Fagin’s Top-k Family of 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 $\rightarrow O(k)$

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 $\rightarrow O(m*n + k)$

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 $\rightarrow O(m*n + k)$
Threshold Algorithm

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 $L_i$;
$\text{high}_i := s(t_i, d)$;
if $d \not\in \text{top-k}$ then {
    look up $s_\nu(d)$ in all lists $L_\nu$ with $\nu \neq i$;
    $\text{score}(d) := \text{aggr} \{s_\nu(d) | \nu=1..m\}$;
    if $\text{score}(d) > \text{min-k}$ then
        add $d$ to top-k and remove min-score $d'$;
        $\text{min-k} := \text{min}\{\text{score}(d') | d' \in \text{top-k}\}$;
threshold := $\text{aggr} \{\text{high}_\nu | \nu=1..m\}$;
if threshold $\leq \text{min-k}$ then exit;
Threshold Algorithm

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

Documents: $d_1, \ldots, d_n$

Threshold algorithm (TA):
scan index lists; consider $d$ at pos $i$ in $L_i$;
$\text{high}_i := s(t_i, d)$;
if $d \not\in \text{top-k}$ then {
   look up $s_{\nu}(d)$ in all lists $L_{\nu}$ with $\nu \neq i$;
   $\text{score}(d) := \text{aggr} \{ s_{\nu}(d) \mid \nu = 1..m \}$;
if $\text{score}(d) > \text{min-k}$ then
   add $d$ to top-k and remove min-score $d'$;
   $\text{min-k} := \text{min} \{ \text{score}(d') \mid d' \in \text{top-k} \}$;
threshold $:= \text{aggr} \{ \text{high}_{\nu} \mid \nu = 1..m \}$;
if threshold $\leq \text{min-k}$ then exit;

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

Posting lists

<table>
<thead>
<tr>
<th>$t_1$</th>
<th>$d78$</th>
<th>$d23$</th>
<th>$d10$</th>
<th>$d1$</th>
<th>$d88$</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_2$</td>
<td>$d64$</td>
<td>$d23$</td>
<td>$d10$</td>
<td>$d12$</td>
<td>$d78$</td>
<td>...</td>
</tr>
<tr>
<td>$t_3$</td>
<td>$d10$</td>
<td>$d78$</td>
<td>$d64$</td>
<td>$d99$</td>
<td>$d34$</td>
<td>...</td>
</tr>
</tbody>
</table>

$k = 2$

Scan depth 1

<table>
<thead>
<tr>
<th>Rank</th>
<th>Doc</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$d78$</td>
<td>0.9</td>
</tr>
<tr>
<td>2</td>
<td>$d64$</td>
<td>0.9</td>
</tr>
</tbody>
</table>
Threshold Algorithm

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

Documents: $d_1, \ldots, d_n$

Threshold algorithm (TA):
scan index lists; consider $d$ at pos $i$ in $L_i$;
$\text{high}_i := s(t_i, d)$;
if $d \notin \text{top-k}$ then {
  look up $s_\nu(d)$ in all lists $L_\nu$ with $\nu \neq i$;
  $\text{score}(d) := \text{aggr} \{s_\nu(d) \mid \nu = 1..m\}$;
  if $\text{score}(d) > \text{min-k}$ then
    add $d$ to top-k and remove min-score $d'$;
    $\text{min-k} := \min\{\text{score}(d') \mid d' \in \text{top-k}\}$;
  threshold := $\text{aggr} \{\text{high}_\nu \mid \nu = 1..m\}$;
  if threshold $\leq \text{min-k}$ then exit;
}

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

Posting lists

<table>
<thead>
<tr>
<th></th>
<th>$d_78$</th>
<th>$d_{23}$</th>
<th>$d_{10}$</th>
<th>$d_1$</th>
<th>$d_{88}$</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_1$</td>
<td>0.9</td>
<td>0.8</td>
<td>0.8</td>
<td>0.7</td>
<td>0.2</td>
<td>...</td>
</tr>
<tr>
<td>$t_2$</td>
<td>0.9</td>
<td>0.6</td>
<td>0.6</td>
<td>0.2</td>
<td>0.1</td>
<td>...</td>
</tr>
<tr>
<td>$t_3$</td>
<td>0.7</td>
<td>0.5</td>
<td>0.3</td>
<td>0.2</td>
<td>0.1</td>
<td>...</td>
</tr>
</tbody>
</table>

$k = 2$

Scan depth 1

<table>
<thead>
<tr>
<th>Rank</th>
<th>Doc</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$d_{78}$</td>
<td>0.9</td>
</tr>
<tr>
<td>2</td>
<td>$d_{64}$</td>
<td>0.9</td>
</tr>
</tbody>
</table>
Threshold Algorithm

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

Documents: $d_1, \ldots, d_n$

Threshold algorithm (TA):
scan index lists; consider $d$ at pos $i$ in $L_i$;
$high_i := s(t_i, d)$;
if $d \not\in$ top-k then {
  look up $s_\nu(d)$ in all lists $L_\nu$ with $\nu \neq i$;
  score($d$) := aggr \{ $s_\nu(d)$ \ where $\nu = 1..m$ \};
if score($d$) $> min-k$ then
  add $d$ to top-k and remove min-score $d'$;
$min-k := \min \{ \text{score}(d') \mid d' \in \text{top-k} \}$;
threshold := aggr \{ $high_\nu$ \ where $\nu = 1..m$ \};
if threshold $\leq min-k$ then exit;

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

Posting lists

<table>
<thead>
<tr>
<th></th>
<th>$t_1$</th>
<th></th>
<th>$t_2$</th>
<th></th>
<th>$t_3$</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$d_1$</td>
<td>d78</td>
<td>0.9</td>
<td>d23</td>
<td>0.8</td>
<td>d10</td>
<td>0.8</td>
</tr>
<tr>
<td></td>
<td>d99</td>
<td>0.2</td>
<td>d12</td>
<td>0.6</td>
<td>d34</td>
<td>0.5</td>
</tr>
</tbody>
</table>

$k = 2$

Scan depth 1

Scan

<table>
<thead>
<tr>
<th>Rank</th>
<th>Doc</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>d78</td>
<td>1.5</td>
</tr>
<tr>
<td>2</td>
<td>d64</td>
<td>0.9</td>
</tr>
</tbody>
</table>
**Threshold Algorithm**

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

Documents: $d_1, \ldots, d_n$

**Posting lists**

<table>
<thead>
<tr>
<th>$t_1$</th>
<th>$d_{78}$</th>
<th>$d_{23}$</th>
<th>$d_{10}$</th>
<th>$d_1$</th>
<th>$d_{88}$</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_2$</td>
<td>$d_{64}$</td>
<td>$d_{23}$</td>
<td>$d_{10}$</td>
<td>$d_{12}$</td>
<td>$d_{78}$</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>0.9</td>
<td>0.8</td>
<td>0.8</td>
<td>0.7</td>
<td>0.2</td>
<td></td>
</tr>
<tr>
<td>$t_3$</td>
<td>$d_{10}$</td>
<td>$d_{78}$</td>
<td>$d_{64}$</td>
<td>$d_{99}$</td>
<td>$d_{34}$</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>0.7</td>
<td>0.5</td>
<td>0.3</td>
<td>0.2</td>
<td>0.1</td>
<td></td>
</tr>
</tbody>
</table>

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 $d \not\in$ top-$k$ then {
look up $s_{\nu}(d)$ in all lists $L_\nu$ with $\nu \neq i$;
$score(d) := \text{aggr}\{s_{\nu}(d) \mid \nu = 1..m\}$;
if $score(d) > \text{min-}k$ then
    add $d$ to top-$k$ and remove min-score $d'$;
$\text{min-}k := \min\{score(d') \mid d' \in \text{top-}k\}$;
$threshold := \text{aggr}\{high_\nu \mid \nu = 1..m\}$;
if $threshold \leq \text{min-}k$ then exit;
}

$k = 2$

Scan depth 1

<table>
<thead>
<tr>
<th>Rank</th>
<th>Doc</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$d_{78}$</td>
<td>1.5</td>
</tr>
<tr>
<td>2</td>
<td>$d_{64}$</td>
<td>0.9</td>
</tr>
</tbody>
</table>
Threshold Algorithm

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 $L_i$;
$\text{high}_i := s(t_i, d);$
if $d \not\in \text{top-k}$ then {
    look up $s_v(d)$ in all lists $L_v$ with $v \neq i$;
    $\text{score}(d) := \text{aggr} \{s_v(d) \mid v = 1..m\}$;
if $\text{score}(d) > \text{min-k}$ then
    add $d$ to top-k and remove min-score $d'$;
$\text{min-k} := \min\{\text{score}(d') \mid d' \in \text{top-k}\}$;
$\text{threshold} := \text{aggr} \{\text{high}_v \mid v = 1..m\}$;
if $\text{threshold} \leq \text{min-k}$ then exit;

Posting lists

<table>
<thead>
<tr>
<th>$t_1$</th>
<th>$d_{78}$</th>
<th>0.9</th>
<th>$d_{23}$</th>
<th>0.8</th>
<th>$d_{10}$</th>
<th>0.8</th>
<th>$d_1$</th>
<th>0.7</th>
<th>$d_{88}$</th>
<th>0.2</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_2$</td>
<td>$d_{64}$</td>
<td>0.9</td>
<td>$d_{23}$</td>
<td>0.6</td>
<td>$d_{10}$</td>
<td>0.6</td>
<td>$d_{12}$</td>
<td>0.2</td>
<td>$d_{78}$</td>
<td>0.1</td>
<td>...</td>
</tr>
<tr>
<td>$t_3$</td>
<td>$d_{10}$</td>
<td>0.7</td>
<td>$d_{78}$</td>
<td>0.5</td>
<td>$d_{64}$</td>
<td>0.3</td>
<td>$d_{99}$</td>
<td>0.2</td>
<td>$d_{34}$</td>
<td>0.1</td>
<td>...</td>
</tr>
</tbody>
</table>

$k = 2$

Scan depth 1

<table>
<thead>
<tr>
<th>Rank</th>
<th>Doc</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$d_{78}$</td>
<td>1.5</td>
</tr>
<tr>
<td>2</td>
<td>$d_{64}$</td>
<td>1.2</td>
</tr>
</tbody>
</table>
Threshold Algorithm

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

Documents: $d_1, \ldots, d_n$

\[
\begin{align*}
  s(t_1, d_1) &= 0.7 \\
  \vdots \\
  s(t_m, d_1) &= 0.2
\end{align*}
\]

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 $d \notin$ top-$k$ then {
  look up $s_\nu(d)$ in all lists $L_\nu$ with $\nu \neq i$;
  $score(d) := \text{aggr} \{s_\nu(d) \mid \nu = 1..m\}$;
  if $score(d) > \text{min-k}$ then
    add $d$ to top-$k$ and remove min-score $d'$;
    $\text{min-k} := \min\{score(d') \mid d' \in \text{top-k}\}$;
  threshold := $\text{aggr} \{high_\nu \mid \nu = 1..m\}$;
  if threshold $\leq \text{min-k}$ then exit;

---

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

\[
\begin{array}{cccc}
  t_1 & d_{78} & d_{23} & d_{10} & d_{1} & d_{88} \\
  0.9 & 0.8 & 0.8 & 0.7 & 0.2 & \ldots \\
  t_2 & d_{64} & d_{23} & d_{10} & d_{12} & d_{78} \\
  0.9 & 0.6 & 0.6 & 0.2 & 0.1 & \ldots \\
  t_3 & d_{10} & d_{78} & d_{64} & d_{99} & d_{34} \\
  0.7 & 0.5 & 0.3 & 0.2 & 0.1 & \ldots \\
\end{array}
\]

\[k = 2\]

Scan depth 1

<table>
<thead>
<tr>
<th>Rank</th>
<th>Doc</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>d_{78}</td>
<td>1.5</td>
</tr>
<tr>
<td>2</td>
<td>d_{64}</td>
<td>1.2</td>
</tr>
</tbody>
</table>
Threshold Algorithm

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

Documents: $d_1, \ldots, d_n$

Threshold algorithm (TA):
scan index lists; consider $d$ at pos $i$ in $L_i$;
$\text{high}_i := s(t_i,d)$;
if $d \notin \text{top-k}$ then {
    look up $s_\nu(d)$ in all lists $L_\nu$ with $\nu \neq i$;
    $\text{score}(d) := \text{aggr} \{s_\nu(d) \mid \nu = 1..m\}$;
    if $\text{score}(d) > \text{min-k}$ then
        add $d$ to top-k and remove min-score $d'$;
    $\text{min-k} := \min\{\text{score}(d') \mid d' \in \text{top-k}\}$;
    $\text{threshold} := \text{aggr}\{\text{high}_\nu \mid \nu = 1..m\}$;
    if $\text{threshold} \leq \text{min-k}$ then exit;
}

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

Posting lists

| $t_1$ | $d_{78}$ | $d_{23}$ | $d_{10}$ | $d_1$ | $d_{88}$ | ...
|-------|----------|----------|----------|-------|----------|------
| $t_2$ | $d_{64}$ | $d_{23}$ | $d_{10}$ | $d_{12}$ | $d_{78}$ | ...
| $t_3$ | $d_{10}$ | $d_{78}$ | $d_{64}$ | $d_{99}$ | $d_{34}$ | ...

$k = 2$

Scan depth 1

<table>
<thead>
<tr>
<th>Rank</th>
<th>Doc</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$d_{10}$</td>
<td>2.1</td>
</tr>
<tr>
<td>2</td>
<td>$d_{78}$</td>
<td>1.5</td>
</tr>
</tbody>
</table>
Threshold Algorithm

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

Documents: $d_1, ..., d_n$

Threshold algorithm (TA):
scan index lists; consider $d$ at pos $i$ in $L_i$;
$\text{high}_i := s(t_i, d)$;
if $d \notin \text{top-k}$ then {
look up $s_{\nu}(d)$ in all lists $L_\nu$ with $\nu \neq i$;
$\text{score}(d) := \text{aggr}\{s_{\nu}(d) | \nu=1..m\}$;
if $\text{score}(d) > \text{min-k}$ then
add $d$ to top-k and remove min-score $d'$;
$\text{min-k} := \text{min}\{\text{score}(d') | d' \in \text{top-k}\}$;
$\text{threshold} := \text{aggr}\{\text{high}_{\nu} | \nu=1..m\}$;
if $\text{threshold} \leq \text{min-k}$ then exit;

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

Posting lists

<table>
<thead>
<tr>
<th></th>
<th>$t_1$</th>
<th>$t_2$</th>
<th>$t_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$d_{78}$</td>
<td>$d_{23}$</td>
<td>$d_{10}$</td>
</tr>
<tr>
<td>$t_1$</td>
<td>0.9</td>
<td>0.8</td>
<td>0.8</td>
</tr>
<tr>
<td>$t_2$</td>
<td>0.9</td>
<td>0.6</td>
<td>0.6</td>
</tr>
<tr>
<td>$t_3$</td>
<td>0.7</td>
<td>0.5</td>
<td>0.3</td>
</tr>
</tbody>
</table>

$k = 2$

Scan depth 2

<table>
<thead>
<tr>
<th>Rank</th>
<th>Doc</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$d_{10}$</td>
<td>2.1</td>
</tr>
<tr>
<td>2</td>
<td>$d_{78}$</td>
<td>1.5</td>
</tr>
</tbody>
</table>

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

Needs only $O(k)$ memory
Threshold Algorithm

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

Documents: \( d_1, \ldots, d_n \)

Threshold algorithm (TA):
scan index lists; consider \( d \) at pos \( i \) in \( L_i \);
\( \text{high}_i := s(t_i, d) \);
if \( d \notin \text{top-k} \) then {
    look up \( s_\nu(d) \) in all lists \( L_\nu \) with \( \nu \neq i \);
    score(d) := aggr \{ s_\nu(d) | \nu = 1..m \};
    if score(d) > min-k then
        add \( d \) to \( \text{top-k} \) and remove min-score \( d' \);
        min-k := min\{score(d') | d' \in \text{top-k}\};
    threshold := aggr \{ \text{high}_\nu | \nu = 1..m \};
    if threshold \leq \text{min-k} \) then exit;

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

Posting lists

\[ \begin{array}{cccccc}
   & d_1 & d_2 & d_3 & d_4 & \ldots \\
\hline
   t_1 & d_{78} & 0.9 & d_{23} & 0.8 & d_{10} & 0.8 & d_1 & 0.7 & d_{88} & 0.2 & \ldots \\
   t_2 & d_{64} & 0.9 & d_{23} & 0.6 & d_{10} & 0.6 & d_{12} & 0.2 & d_{78} & 0.1 & \ldots \\
   t_3 & d_{10} & 0.7 & d_{78} & 0.5 & d_{64} & 0.3 & d_{99} & 0.2 & d_{34} & 0.1 & \ldots \\
\end{array} \]

Rank    Doc   Score
1       d_{10}  2.1
2       d_{78}  1.5

Scan depth 3

k = 2
Threshold Algorithm

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

Documents: $d_1, ..., d_n$

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

Threshold algorithm (TA):
scan index lists; consider $d$ at pos $i$ in $L_i$;
$\text{high}_i := s(t_i, d)$;
if $d \notin \text{top-k}$ then {
    look up $s_\nu(d)$ in all lists $L_\nu$ with $\nu \neq i$;
    score($d$) := aggr $\{s_\nu(d) | \nu = 1..m\}$;
    if score($d$) > $\text{min-k}$ then
        add $d$ to top-k and remove min-score $d'$;
    $\text{min-k} := \min\{\text{score}(d') | d' \in \text{top-k}\}$;
    threshold := aggr $\{\text{high}_\nu | \nu = 1..m\}$;
    if threshold $\leq \text{min-k}$ then exit;

\begin{tabular}{|c|c|c|c|c|c|}
\hline
$t_1$ & $d_78$ & $d_{23}$ & $d_{10}$ & $d_1$ & $d_{88}$ \\
0.9 & 0.8 & 0.8 & 0.7 & 0.2 & ... \\
$\text{high}_i := s(t_i, d)$ \\
\hline
\end{tabular}

\begin{tabular}{|c|c|c|c|c|c|}
\hline
$t_2$ & $d_{64}$ & $d_{23}$ & $d_{10}$ & $d_{12}$ & $d_{78}$ \\
0.9 & 0.6 & 0.6 & 0.2 & 0.1 & ... \\
\hline
\end{tabular}

\begin{tabular}{|c|c|c|c|c|c|}
\hline
$t_3$ & $d_{10}$ & $d_{78}$ & $d_{64}$ & $d_{99}$ & $d_{34}$ \\
0.7 & 0.5 & 0.3 & 0.2 & 0.1 & ... \\
\hline
\end{tabular}

$k = 2$

Scan depth 3

\begin{tabular}{|c|c|c|}
\hline
Rank & Doc & Score \\
\hline
1 & $d_{10}$ & 2.1 \\
2 & $d_{78}$ & 1.5 \\
\hline
\end{tabular}
Threshold Algorithm

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

Documents: $d_1$, ..., $d_n$

<table>
<thead>
<tr>
<th>Posting lists</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_1$</td>
</tr>
<tr>
<td>$t_2$</td>
</tr>
<tr>
<td>$t_3$</td>
</tr>
</tbody>
</table>

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

Threshold algorithm (TA):
scan index lists; consider $d$ at pos $i$ in $L_i$;

$\text{high}_i := s(t_i,d)$;
if $d \notin \text{top-k}$ then {
    look up $s_\nu(d)$ in all lists $L_\nu$ with $\nu \neq i$;
    $\text{score}(d) := \text{aggr} \{ s_\nu(d) | \nu = 1..m \}$;
if $\text{score}(d) > \min-k$ then
    add $d$ to top-k and remove min-score $d'$;
    $\min-k := \min \{ \text{score}(d') | d' \in \text{top-k} \}$;
threshold := $\text{aggr} \{ \text{high}_\nu | \nu = 1..m \}$;
if threshold $\leq \min-k$ then exit;
Threshold Algorithm

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

Documents: $d_1, \ldots, d_n$

Threshold algorithm (TA):
scan index lists; consider $d$ at pos $i$ in $L_i$;
$\text{high}_i := s(t_i, d)$;
if $d \not\in \text{top-}k$ then {
    look up $s_\nu(d)$ in all lists $L_\nu$ with $\nu \neq i$;
    $\text{score}(d) := \text{aggr} \{s_\nu(d) \mid \nu = 1..m\}$;
    if $\text{score}(d) > \text{min-}k$ then
        add $d$ to top-$k$ and remove min-score $d'$;
        $\text{min-}k := \min\{\text{score}(d') \mid d' \in \text{top-}k\}$;
    threshold := $\text{aggr} \{\text{high}_\nu \mid \nu = 1..m\}$;
    if threshold $\leq \text{min-}k$ then exit;
}

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

Posting lists

$k = 2$

Scan depth 4

Threshold algorithm (TA):
scan index lists; consider $d$ at pos $i$ in $L_i$;
$\text{high}_i := s(t_i, d)$;
if $d \not\in \text{top-}k$ then {
    look up $s_\nu(d)$ in all lists $L_\nu$ with $\nu \neq i$;
    $\text{score}(d) := \text{aggr} \{s_\nu(d) \mid \nu = 1..m\}$;
    if $\text{score}(d) > \text{min-}k$ then
        add $d$ to top-$k$ and remove min-score $d'$;
        $\text{min-}k := \min\{\text{score}(d') \mid d' \in \text{top-}k\}$;
    threshold := $\text{aggr} \{\text{high}_\nu \mid \nu = 1..m\}$;
    if threshold $\leq \text{min-}k$ then exit;
Threshold Algorithm

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

Documents: $d_1, ..., d_n$

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

Threshold algorithm (TA):
scan index lists; consider $d$ at pos $i$ in $L_i$;

$\text{high}_i := s(t_i, d)$;

if $d \notin \text{top-k}$ then {
  look up $s_\nu(d)$ in all lists $L_\nu$ with $\nu \neq i$;
  $\text{score}(d) := \text{aggr} \{s_\nu(d) | \nu = 1..m\}$;

  if $\text{score}(d) > \text{min-k}$ then
    add $d$ to top-$k$ and remove min-score $d'$;

  $\text{min-k} := \min\{\text{score}(d') | d' \in \text{top-k}\}$;

  $\text{threshold} := \text{aggr} \{\text{high}_\nu | \nu = 1..m\}$;

  if $\text{threshold} \leq \text{min-k}$ then exit;
}

Threshold Algorithm

Posting lists

<table>
<thead>
<tr>
<th>$t_1$</th>
<th>$d_{78}$</th>
<th>$d_{23}$</th>
<th>$d_{10}$</th>
<th>$d_{11}$</th>
<th>$d_{88}$</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_2$</td>
<td>$d_{64}$</td>
<td>$d_{23}$</td>
<td>$d_{10}$</td>
<td>$d_{12}$</td>
<td>$d_{78}$</td>
<td>...</td>
</tr>
<tr>
<td>$t_3$</td>
<td>$d_{10}$</td>
<td>$d_{78}$</td>
<td>$d_{64}$</td>
<td>$d_{99}$</td>
<td>$d_{34}$</td>
<td>...</td>
</tr>
</tbody>
</table>

$k = 2$

Scan depth 4

Rank    Doc   Score

1          d_{10}     2.1
2          d_{78}     1.5

STOP!
No-Random-Access Algorithm

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

Documents: \(d_1, \ldots, d_n\)

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

**No-Random-Access algorithm (NRA):**
scan index lists; consider \(d\) at pos \(i\) in \(L_i\);

\[
E(d) := E(d) \cup \{i\}; \quad \text{high}_i := s(t_i, d);
\]

\[
\text{worstscore}(d) := \text{aggr}\{s(t_\nu, d) \mid \nu \in E(d)\};
\]

\[
\text{bestscore}(d) := \text{aggr}\{\text{worstscore}(d), \text{aggr}\{\text{high}_\nu \mid \nu \notin E(d)\}\};
\]

if \(\text{worstscore}(d) > \text{min}-k\) then add \(d\) to top-\(k\)

\[
\text{min}-k := \min\{\text{worstscore}(d') \mid d' \in \text{top}-k\};
\]

else if \(\text{bestscore}(d) > \text{min}-k\) then

\[
\text{cand} := \text{cand} \cup \{d\};
\]

\[
\text{threshold} := \max\{\text{bestscore}(d') \mid d' \in \text{cand}\};
\]

if \(\text{threshold} \leq \text{min}-k\) then exit;
No-Random-Access Algorithm

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

Documents: \( d_1, \ldots, d_n \)

\[
\begin{array}{c|ccccc}
\hline
\text{Index lists} & d78 & d23 & d10 & d1 & d88 \\
\hline
\text{depth 1} & 0.9 & 0.8 & 0.8 & 0.7 & 0.2 \\
\hline
\text{depth 2} & d64 & d23 & d10 & d12 & d78 \\
\hline
\text{depth 3} & d10 & d78 & d64 & d99 & d34 \\
\hline
\end{array}
\]

\[
\begin{array}{c|c|c|c}
\hline
\text{Rank} & \text{Doc} & \text{Worst-score} & \text{Best-score} \\
\hline
1 & d78 & 0.9 & 2.4 \\
\hline
2 & d64 & 0.8 & 2.4 \\
\hline
3 & d10 & 0.7 & 2.4 \\
\hline
\end{array}
\]

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

Scan depth 1
No-Random-Access Algorithm (NRA):
scan index lists; consider \( d \) at pos \( i \) in \( L_i \);
\( E(d) := E(d) \cup \{i\} \); \( \text{high}_i := s(t_i,d) \);
\( \text{worstscore}(d) := \text{aggr}\{s(t_\nu,d) \mid \nu \in E(d)\} \);
\( \text{bestscore}(d) := \text{aggr}\{\text{worstscore}(d), \text{aggr}\{\text{high}_\nu \mid \nu \notin E(d)\}\} \};

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

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

Documents: \( d_1, \ldots, d_n \)

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

Index lists

<table>
<thead>
<tr>
<th>( t_1 )</th>
<th>( d_{78} )</th>
<th>( d_{23} )</th>
<th>( d_{10} )</th>
<th>( d_1 )</th>
<th>( d_{88} )</th>
<th>( \ldots )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.9</td>
<td>0.8</td>
<td>0.8</td>
<td>0.7</td>
<td>0.2</td>
<td>\ldots</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( t_2 )</th>
<th>( d_{64} )</th>
<th>( d_{23} )</th>
<th>( d_{10} )</th>
<th>( d_{12} )</th>
<th>( d_{78} )</th>
<th>( \ldots )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.8</td>
<td>0.6</td>
<td>0.6</td>
<td>0.2</td>
<td>0.1</td>
<td>\ldots</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( t_3 )</th>
<th>( d_{10} )</th>
<th>( d_{78} )</th>
<th>( d_{64} )</th>
<th>( d_{99} )</th>
<th>( d_{34} )</th>
<th>( \ldots )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.7</td>
<td>0.5</td>
<td>0.4</td>
<td>0.2</td>
<td>0.1</td>
<td>\ldots</td>
</tr>
</tbody>
</table>

\( k = 1 \)

Scan depth 2

<table>
<thead>
<tr>
<th>Rank</th>
<th>Doc</th>
<th>Worst-score</th>
<th>Best-score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>d_{78}</td>
<td>1.4</td>
<td>2.0</td>
</tr>
<tr>
<td>2</td>
<td>d_{23}</td>
<td>1.4</td>
<td>1.9</td>
</tr>
<tr>
<td>3</td>
<td>d_{64}</td>
<td>0.8</td>
<td>2.1</td>
</tr>
<tr>
<td>4</td>
<td>d_{10}</td>
<td>0.7</td>
<td>2.1</td>
</tr>
</tbody>
</table>

Avishek Anand
No-Random-Access Algorithm

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

Documents: \( d_1, \ldots, d_n \)

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\(_i := s(t_i,d); \)
\( \text{worstscore}(d) := \text{aggr}\{s(t_\nu,d) \mid \nu \in E(d)\}; \)
\( \text{bestscore}(d) := \text{aggr}\{\text{worstscore}(d), \)
\text{aggr}\{\text{high}_\nu \mid \nu \notin E(d)\}\}; \)
if worstscore\(_d > \min\_k \) then add \( d \) to top-\( k \)
\( \min\_k := \min\{\text{worstscore}(d') \mid d' \in \text{top-}k\}; \)
else if bestscore\(_d > \min\_k \) then
\( \text{cand} := \text{cand} \cup \{d\}; \)
threshold := max \{bestscore\(_{d'} \mid d' \in \text{cand}\}; \)
if threshold \( \leq \min\_k \) then exit;

Index lists

\[
\begin{array}{c|cccccc}
| t_1 | d78 & d23 & d10 & d1 & d88 & \ldots \\
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.9</td>
<td>0.8</td>
<td>0.8</td>
<td>0.7</td>
<td>0.2</td>
</tr>
</tbody>
</table>
\end{array}
\]

\[
\begin{array}{c|cccccc}
| t_2 | d64 & d23 & d10 & d12 & d78 & \ldots \\
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.8</td>
<td>0.6</td>
<td>0.6</td>
<td>0.2</td>
<td>0.1</td>
</tr>
</tbody>
</table>
\end{array}
\]

\[
\begin{array}{c|cccccc}
| t_3 | d10 & d78 & d64 & d99 & d34 & \ldots \\
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.7</td>
<td>0.5</td>
<td>0.4</td>
<td>0.2</td>
<td>0.1</td>
</tr>
</tbody>
</table>
\end{array}
\]

STOP!
Combined Algorithm

Balanced SA/RA Scheduling:

Define cost ratio \( \frac{C_{\text{RA}}}{C_{\text{SA}}} =: r \)

(e.g., based on statistics for execution environment (“middleware”),
typical values: \( \frac{C_{\text{RA}}}{C_{\text{SA}}} \approx 20-10,000 \) for a hard-disk)

Perform \textbf{NRA} (using sorted access only)

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

Cost \textbf{competitiveness} w.r.t. “optimal schedule”

(\text{scan until } \sum_i \text{high}_i \leq \min\{\text{bestscore}(d) \mid d \in \text{final top-k}\},
then perform RAs for all \( d' \) with \( \text{bestscore}(d') > \min\text{-k} \): \( 4m + k \)
# Combined Algorithm

<table>
<thead>
<tr>
<th></th>
<th><strong>L₁</strong></th>
<th></th>
<th><strong>L₂</strong></th>
<th></th>
<th><strong>L₃</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0.8</td>
<td></td>
<td>G</td>
<td>0.7</td>
<td>Y</td>
<td>0.9</td>
</tr>
<tr>
<td>B</td>
<td>0.2</td>
<td></td>
<td>H</td>
<td>0.5</td>
<td>A</td>
<td>0.7</td>
</tr>
<tr>
<td>K</td>
<td>0.19</td>
<td></td>
<td>R</td>
<td>0.5</td>
<td>P</td>
<td>0.3</td>
</tr>
<tr>
<td>F</td>
<td>0.17</td>
<td></td>
<td>Y</td>
<td>0.5</td>
<td>F</td>
<td>0.25</td>
</tr>
<tr>
<td>M</td>
<td>0.16</td>
<td></td>
<td>W</td>
<td>0.3</td>
<td>S</td>
<td>0.25</td>
</tr>
<tr>
<td>Z</td>
<td>0.15</td>
<td></td>
<td>D</td>
<td>0.25</td>
<td>T</td>
<td>0.2</td>
</tr>
<tr>
<td>W</td>
<td>0.1</td>
<td></td>
<td>W</td>
<td>0.2</td>
<td>Q</td>
<td>0.15</td>
</tr>
<tr>
<td>Q</td>
<td>0.07</td>
<td></td>
<td>A</td>
<td>0.2</td>
<td>X</td>
<td>0.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Combined Algorithm**

<table>
<thead>
<tr>
<th></th>
<th>$L_1$</th>
<th>$L_2$</th>
<th>$L_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0.8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>0.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>K</td>
<td>0.19</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>0.17</td>
<td></td>
<td></td>
</tr>
<tr>
<td>M</td>
<td>0.16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Z</td>
<td>0.15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>W</td>
<td>0.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q</td>
<td>0.07</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

...  

**CA:** compute top-1 result using one RA after every round of SA
## Combined Algorithm

<table>
<thead>
<tr>
<th></th>
<th>L₁</th>
<th>L₂</th>
<th>L₃</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0.8</td>
<td>G</td>
<td>0.7</td>
</tr>
<tr>
<td>B</td>
<td>0.2</td>
<td>H</td>
<td>0.5</td>
</tr>
<tr>
<td>K</td>
<td>0.19</td>
<td>R</td>
<td>0.5</td>
</tr>
<tr>
<td>F</td>
<td>0.17</td>
<td>Y</td>
<td>0.5</td>
</tr>
<tr>
<td>M</td>
<td>0.16</td>
<td>W</td>
<td>0.3</td>
</tr>
<tr>
<td>Z</td>
<td>0.15</td>
<td>D</td>
<td>0.25</td>
</tr>
<tr>
<td>W</td>
<td>0.1</td>
<td>W</td>
<td>0.2</td>
</tr>
<tr>
<td>Q</td>
<td>0.07</td>
<td>A</td>
<td>0.2</td>
</tr>
<tr>
<td>..</td>
<td>..</td>
<td>..</td>
<td>..</td>
</tr>
</tbody>
</table>
## Combined Algorithm

<table>
<thead>
<tr>
<th></th>
<th>L₁</th>
<th>L₂</th>
<th>L₃</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0.8</td>
<td></td>
<td>Y: 0.9</td>
</tr>
<tr>
<td>B</td>
<td>0.2</td>
<td>H: 0.5</td>
<td>A: 0.7</td>
</tr>
<tr>
<td>K</td>
<td>0.19</td>
<td>R: 0.5</td>
<td>P: 0.3</td>
</tr>
<tr>
<td>F</td>
<td>0.17</td>
<td>Y: 0.5</td>
<td>F: 0.25</td>
</tr>
<tr>
<td>M</td>
<td>0.16</td>
<td>W: 0.3</td>
<td>S: 0.25</td>
</tr>
<tr>
<td>Z</td>
<td>0.15</td>
<td>D: 0.25</td>
<td>T: 0.2</td>
</tr>
<tr>
<td>W</td>
<td>0.1</td>
<td>W: 0.2</td>
<td>Q: 0.15</td>
</tr>
<tr>
<td>Q</td>
<td>0.07</td>
<td>A: 0.2</td>
<td>X: 0.1</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

### Candidates:
- A: [0.8, 2.4]
- Y: [0.9, 2.4]
- G: [0.7, 2.4]
- ?: [0.0, 2.4]
### Combined Algorithm

<table>
<thead>
<tr>
<th>L₁</th>
<th>L₂</th>
<th>L₃</th>
</tr>
</thead>
<tbody>
<tr>
<td>A: 0.8</td>
<td>G: 0.7</td>
<td>Y: 0.9</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Candidates</th>
</tr>
</thead>
<tbody>
<tr>
<td>A: [0.8, 2.4]</td>
</tr>
<tr>
<td>G: [0.7, 2.4]</td>
</tr>
<tr>
<td>Y: [0.9, 2.4]</td>
</tr>
<tr>
<td>?: [0.0, 2.4]</td>
</tr>
</tbody>
</table>
Avishek Anand

Combined Algorithm

L₁ | L₂ | L₃
---|---|---
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

1️⃣ 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: [1.7, 1.7]  Y: [0.9, 2.4]
G: [0.7, 2.4]  ?: [0.0, 2.4]
Combined Algorithm

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: [1.7, 1.7]  Y: [0.9, 1.6]
G: [0.7, 1.6]  ?: [0.0, 1.4]
Combined Algorithm

<table>
<thead>
<tr>
<th></th>
<th>L₁</th>
<th>L₂</th>
<th>L₃</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0.8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>0.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>K</td>
<td>0.19</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>0.17</td>
<td></td>
<td></td>
</tr>
<tr>
<td>M</td>
<td>0.16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Z</td>
<td>0.15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>W</td>
<td>0.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q</td>
<td>0.07</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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: [1.7, 1.7]
- Y: [0.9, 1.6]
- G: [0.7, 1.6]
- ?: [0.0, 1.4]
## Combined Algorithm

<table>
<thead>
<tr>
<th></th>
<th>$L_1$</th>
<th></th>
<th>$L_2$</th>
<th></th>
<th>$L_3$</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0.8</td>
<td></td>
<td>G</td>
<td>0.7</td>
<td></td>
<td>Y</td>
</tr>
<tr>
<td>B</td>
<td>0.2</td>
<td></td>
<td>H</td>
<td>0.5</td>
<td></td>
<td>A</td>
</tr>
<tr>
<td>K</td>
<td>0.19</td>
<td></td>
<td>R</td>
<td>0.5</td>
<td></td>
<td>P</td>
</tr>
<tr>
<td>F</td>
<td>0.17</td>
<td></td>
<td>Y</td>
<td>0.5</td>
<td></td>
<td>F</td>
</tr>
<tr>
<td>M</td>
<td>0.16</td>
<td></td>
<td>W</td>
<td>0.3</td>
<td></td>
<td>S</td>
</tr>
<tr>
<td>Z</td>
<td>0.15</td>
<td></td>
<td>D</td>
<td>0.25</td>
<td></td>
<td>T</td>
</tr>
<tr>
<td>W</td>
<td>0.1</td>
<td></td>
<td>W</td>
<td>0.2</td>
<td></td>
<td>Q</td>
</tr>
<tr>
<td>Q</td>
<td>0.07</td>
<td></td>
<td>A</td>
<td>0.2</td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

### 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: $[1.7, 1.7]$
- Y: $[0.9, 1.6]$
- G: $[0.7, 1.6]$
- ?: $[0.0, 1.4]$
### Combined Algorithm

<table>
<thead>
<tr>
<th>$L_1$</th>
<th>$L_2$</th>
<th>$L_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>A: 0.8</td>
<td>G: 0.7</td>
<td>Y: 0.9</td>
</tr>
<tr>
<td>B: 0.2</td>
<td>H: 0.5</td>
<td>A: 0.7</td>
</tr>
<tr>
<td>K: 0.19</td>
<td>R: 0.5</td>
<td>P: 0.3</td>
</tr>
<tr>
<td>F: 0.17</td>
<td>Y: 0.5</td>
<td>F: 0.25</td>
</tr>
<tr>
<td>M: 0.16</td>
<td>W: 0.3</td>
<td>S: 0.25</td>
</tr>
<tr>
<td>Z: 0.15</td>
<td>D: 0.25</td>
<td>T: 0.2</td>
</tr>
<tr>
<td>W: 0.1</td>
<td>W: 0.2</td>
<td>Q: 0.15</td>
</tr>
<tr>
<td>Q: 0.07</td>
<td>A: 0.2</td>
<td>X: 0.1</td>
</tr>
<tr>
<td>:</td>
<td>:</td>
<td>:</td>
</tr>
</tbody>
</table>

#### 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: [1.7, 1.7]
- Y: [0.9, 1.6]
- G: [0.7, 1.6]
- ?: [0.9, 1.4]
### Combined Algorithm

<table>
<thead>
<tr>
<th></th>
<th>$L_1$</th>
<th>$L_2$</th>
<th>$L_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0.8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>0.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>K</td>
<td>0.19</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>0.17</td>
<td></td>
<td></td>
</tr>
<tr>
<td>M</td>
<td>0.16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Z</td>
<td>0.15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>W</td>
<td>0.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q</td>
<td>0.07</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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: [1.7, 1.7]
- Y: [0.9, 1.6]
- G: [0.7, 1.6]
- ?: [0.9, 1.4]
## Combined Algorithm

<table>
<thead>
<tr>
<th></th>
<th>L₁</th>
<th>L₂</th>
<th>L₃</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0.8</td>
<td>G: 0.7</td>
<td>Y: 0.9</td>
</tr>
<tr>
<td>B</td>
<td>0.2</td>
<td>H: 0.5</td>
<td>A: 0.7</td>
</tr>
</tbody>
</table>

**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.

**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: [1.7, 1.7]
- G: [0.7, 1.6]
- Y: [0.9, 1.6]
- ?: [0.9, 1.4]
Combined Algorithm

1<sup>st</sup> 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.

2<sup>nd</sup> 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!

execution costs: 6 SA + 1 RA

candidates:
- A: [1.7, 1.7]
- Y: [0.9, 1.6]
- G: [0.7, 1.6]
- ?: [0.9, 1.4]
Combined Algorithm

<table>
<thead>
<tr>
<th></th>
<th>L₁</th>
<th>L₂</th>
<th>L₃</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0.8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>0.2</td>
<td></td>
<td>A: 0.7</td>
</tr>
<tr>
<td>K</td>
<td>0.19</td>
<td></td>
<td>P: 0.3</td>
</tr>
<tr>
<td>F</td>
<td>0.17</td>
<td>R: 0.5</td>
<td>F: 0.25</td>
</tr>
<tr>
<td>M</td>
<td>0.16</td>
<td>W: 0.2</td>
<td>S: 0.25</td>
</tr>
</tbody>
</table>

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.

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!

execution costs: 6 SA + 1 RA

candidates:
- A: [1.7, 1.7]
- Y: [0.9, 1.6]
- G: [0.7, 1.6]
- ?: [0.9, 1.4]
What is the cost (in terms of p and c) of finding top-1 results using:

- TA
- NRA

For what values of p does TA/NRA win?

For p = 3, how many rounds does CA take? ***

Simple Computational Model

cost of sequential access = c
cost of random access = p.c

<table>
<thead>
<tr>
<th></th>
<th>L1</th>
<th>L2</th>
<th>L3</th>
</tr>
</thead>
<tbody>
<tr>
<td>doc 25</td>
<td>0.6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>doc 17</td>
<td>0.6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>doc 83</td>
<td>0.9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>doc 78</td>
<td>0.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>doc 38</td>
<td>0.6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>doc 17</td>
<td>0.7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>doc 83</td>
<td>0.4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>doc 14</td>
<td>0.6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>doc 61</td>
<td>0.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>doc 17</td>
<td>0.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>doc 5</td>
<td>0.6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>doc 81</td>
<td>0.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>doc 21</td>
<td>0.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>doc 83</td>
<td>0.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>doc 65</td>
<td>0.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>doc 91</td>
<td>0.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>doc 21</td>
<td>0.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>doc 10</td>
<td>0.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>doc 44</td>
<td>0.1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
References and Further Readings

http://www.ir.uwaterloo.ca/book/

http://www.cis.upenn.edu/~jstoy/cis650/papers/WAND.pdf


credits: Gerhard Weikum for IRDM 2009
Further Reading

- C. Buckley, A. F. Lewit: Optimization of Inverted Vector Searches. SIGIR 1985
- Martin Theobald, Ralf Schenkel, Gerhard Weikum: Efficient and self-tuning incremental query expansion for top-k query processing. SIGIR 2005