V.3 Query Processing

1. Term-at-a-Time
2. Document-at-a-Time
3. WAND
4. Quit & Continue
5. Buckley’s Algorithm
6. Fagin’s Threshold Algorithms
7. Query Processing with Importance Scores
8. Query Processing with Champion Lists

Based on MRS Chapter 7 and RBY Chapter 9
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., \(tf.idf_t,d\))
Inverted Index

- Document-ordered or score-ordered posting lists
- Posting lists with skip pointers allow for faster traversal
Overview of Query Processing Methods

- **Holistic** query processing methods determine whole query result
  - Term-at-a-Time
  - Document-at-a-Time

- **Top-$k$** query processing methods determine top-$k$ query result
  - WAND
  - Quit & Continue
  - Fagin’s Threshold Algorithms

- Opportunities for optimization over **naïve merge & sort baseline**
  - **skipping** in document-ordered posting lists
  - **early termination** of query processing for score-ordered posting lists
1. Term-at-a-Time Query Processing

- **Term-at-a-Time (TAAT) query processing**
  - reads posting lists for query terms \( \langle t_1, \ldots, t_{|q|} \rangle \) **successively**
  - maintains an **accumulator** for each result document with value

\[
\text{acc}(d) = \sum_{i \leq j} \text{score}(t_i, d) \quad \text{after the first } j \text{ posting lists have been read}
\]

- **required memory** depends on the **number of accumulators** maintained

- **top-\(k\) results** can be determined by **sorting accumulators** at the end
1. Term-at-a-Time Query Processing

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

  • reads posting lists for query terms $\langle t_1, \ldots, t_{|q|} \rangle$ **successively**

  • maintains an **accumulator** for each result document with value

\[
\text{acc}(d) = \sum_{i \leq j} \text{score}(t_i, d) \text{ after the first } j \text{ posting lists have been read}
\]

  \[a \ldots \quad d_1, 1.0 \quad d_4, 2.0 \quad d_7, 0.2 \quad d_8, 0.1\]

  \[b \ldots \quad d_4, 1.0 \quad d_7, 2.0 \quad d_8, 0.2 \quad d_9, 0.1\]

  \[c \ldots \quad d_4, 3.0 \quad d_7, 1.0\]

  \[\text{Accumulators}\]

  \[d_1 : 0.0\]

  \[d_4 : 0.0\]

  \[d_7 : 0.0\]

  \[d_8 : 0.0\]

  \[d_9 : 0.0\]

• required **memory** depends on the **number of accumulators** maintained

• **top-k results** can be determined by **sorting accumulators** at the end
1. Term-at-a-Time Query Processing

- **Term-at-a-Time (TAAT)** query processing
  - reads posting lists for query terms \( \langle t_1, \ldots, t_{|q|} \rangle \) **successively**
  - maintains an **accumulator** for each result document with value

\[
acc(d) = \sum_{i \leq j} score(t_i, d) \text{ after the first } j \text{ posting lists have been read}
\]

<table>
<thead>
<tr>
<th>Accumulators</th>
</tr>
</thead>
<tbody>
<tr>
<td>(d_1) : 1.0</td>
</tr>
<tr>
<td>(d_4) : 0.0</td>
</tr>
<tr>
<td>(d_7) : 0.0</td>
</tr>
<tr>
<td>(d_8) : 0.0</td>
</tr>
<tr>
<td>(d_9) : 0.0</td>
</tr>
</tbody>
</table>

- required **memory** depends on the **number of accumulators** maintained

- **top-\(k\) results** can be determined by **sorting accumulators** at the end
1. Term-at-a-Time Query Processing

- **Term-at-a-Time** (TAAT) query processing
  - reads posting lists for query terms \( \langle t_1, \ldots, t_{|q|} \rangle \) **successively**
  - maintains an **accumulator** for each result document with value
    \[
    acc(d) = \sum_{i \leq j} score(t_i, d)
    \]
    after the first \( j \) posting lists have been read

\[
\begin{align*}
    a & \quad \cdots \quad d_1, 1.0 \quad d_4, 2.0 \quad d_7, 0.2 \quad d_8, 0.1 \\
    b & \quad \cdots \quad d_4, 1.0 \quad d_7, 2.0 \quad d_8, 0.2 \quad d_9, 0.1 \\
    c & \quad \cdots \quad d_4, 3.0 \quad d_7, 1.0
\end{align*}
\]

- required **memory** depends on the **number of accumulators** maintained
- **top-\( k \) results** can be determined by **sorting accumulators** at the end
1. Term-at-a-Time Query Processing

- **Term-at-a-Time** (TAAT) query processing
  - reads posting lists for query terms \( \langle t_1, \ldots, t_{|q|} \rangle \) **successively**
  - maintains an **accumulator** for each result document with value

\[
acc(d) = \sum_{i \leq j} score(t_i, d) \tag{1}
\]

after the first \( j \) posting lists have been read

- **Accumulators**

\[
\begin{array}{cccc}
  a & \cdots & d_1, 1.0 & d_4, 2.0 & d_7, 0.2 & d_8, 0.1 \\
  b & \cdots & d_4, 1.0 & d_7, 2.0 & d_8, 0.2 & d_9, 0.1 \\
  c & \cdots & d_4, 3.0 & d_7, 1.0 \\
\end{array}
\]

- required **memory** depends on the **number of accumulators** maintained
- **top-\( k \) results** can be determined by **sorting accumulators** at the end
1. Term-at-a-Time Query Processing

- **Term-at-a-Time** (TAAT) query processing
- reads posting lists for query terms \(\langle t_1, \ldots, t_{|q|}\rangle\) **successively**
- maintains an **accumulator** for each result document with value

\[
acc(d) = \sum_{i \leq j} score(t_i, d) \quad \text{after the first } j \text{ posting lists have been read}
\]

- required **memory** depends on the **number of accumulators** maintained
- **top-\(k\) results** can be determined by **sorting accumulators** at the end

**Accumulators**

<table>
<thead>
<tr>
<th></th>
<th>(d_1) : 1.0</th>
<th>(d_4) : 2.0</th>
<th>(d_7) : 0.2</th>
<th>(d_8) : 0.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a)</td>
<td>(d_1), 1.0</td>
<td>(d_4), 2.0</td>
<td>(d_7), 0.2</td>
<td>(d_8), 0.1</td>
</tr>
<tr>
<td>(b)</td>
<td>(d_4), 1.0</td>
<td>(d_7), 2.0</td>
<td>(d_8), 0.2</td>
<td>(d_9), 0.1</td>
</tr>
<tr>
<td>(c)</td>
<td>(d_4), 3.0</td>
<td>(d_7), 1.0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\(\text{IR&DM '13/'14}\)
1. Term-at-a-Time Query Processing

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

  - reads posting lists for query terms $\langle t_1, \ldots, t_{|q|} \rangle$ **successively**
  
  - maintains an **accumulator** for each result document with value

$$acc(d) = \sum_{i \leq j} score(t_i, d)$$

  after the first $j$ posting lists have been read

- **required memory** depends on the **number of accumulators** maintained

- **top-$k$ results** can be determined by sorting accumulators at the end

Accumulators

<table>
<thead>
<tr>
<th></th>
<th>$d_1$</th>
<th>$d_4$</th>
<th>$d_7$</th>
<th>$d_8$</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>$d_1$, 1.0</td>
<td>$d_4$, 2.0</td>
<td><strong>$d_7$, 0.2</strong></td>
<td>$d_8$, 0.1</td>
</tr>
<tr>
<td>b</td>
<td>$d_4$, 1.0</td>
<td>$d_7$, 2.0</td>
<td>$d_8$, 0.2</td>
<td>$d_9$, 0.1</td>
</tr>
<tr>
<td>c</td>
<td>$d_4$, 3.0</td>
<td>$d_7$, 1.0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
1. Term-at-a-Time Query Processing

- **Term-at-a-Time** (TAAT) query processing
  - reads posting lists for query terms \( \langle t_1, \ldots, t_{|q|} \rangle \) **successively**
  - maintains an **accumulator** for each result document with value

\[
acc(d) = \sum_{i \leq j} score(t_i, d)
\]

  after the first \( j \) posting lists have been read

  - required **memory** depends on the **number of accumulators** maintained
  - **top-k results** can be determined by **sorting accumulators** at the end
1. Term-at-a-Time Query Processing

- **Term-at-a-Time** (TAAT) query processing
  - reads posting lists for query terms $\langle t_1, \ldots, t_{|q|} \rangle$ successively
  - maintains an **accumulator** for each result document with value $acc(d) = \sum_{i \leq j} score(t_i, d)$ after the first $j$ posting lists have been read

\[
acc(d) = \sum_{i \leq j} score(t_i, d)
\]

- required **memory** depends on the **number of accumulators** maintained
- **top-$k$ results** can be determined by **sorting accumulators** at the end

<table>
<thead>
<tr>
<th>Accumulators</th>
</tr>
</thead>
<tbody>
<tr>
<td>$d_1$ : 1.0</td>
</tr>
<tr>
<td>$d_4$ : 2.0</td>
</tr>
<tr>
<td>$d_7$ : 0.2</td>
</tr>
<tr>
<td>$d_8$ : 0.1</td>
</tr>
</tbody>
</table>

| $a$ .......... |
| $d_1$, 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 |
1. Term-at-a-Time Query Processing

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

  - reads posting lists for query terms \( \langle t_1, \ldots, t_{|q|} \rangle \) **successively**

  - maintains an **accumulator** for each result document with value

    \[
    acc(d) = \sum_{i \leq j} score(t_i, d) \quad \text{after the first } j \text{ posting lists have been read}
    \]


\[
\begin{array}{c|c|c|c|c}
\text{Accumulators} & d_1 & d_4 & d_7 & d_8 \\
\hline
d_1 & 1.0 & & & \\
d_4 & & 2.0 & & \\
d_7 & & & 0.2 & \\
d_8 & & & & 0.1 \\
d_9 & & & 0.2 & \\
\end{array}
\]

- required **memory** depends on the **number of accumulators** maintained

- **top-k results** can be determined by **sorting accumulators** at the end
1. Term-at-a-Time Query Processing

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

  • reads posting lists for query terms \( \langle t_1, \ldots, t_{|q|} \rangle \) **successively**

  • maintains an **accumulator** for each result document with value

\[
acc(d) = \sum_{i \leq j} score(t_i, d) \text{ after the first } j \text{ posting lists have been read}
\]

\[
\begin{align*}
\text{Accumulators} & \\
 a & : d_1: 1.0 \quad d_4: 2.0 \quad d_7: 0.2 \quad d_8: 0.1 \\
 b & : d_4: 1.0 \quad d_7: 2.0 \quad d_8: 0.2 \quad d_9: 0.1 \\
 c & : d_4: 3.0 \quad d_7: 1.0
\end{align*}
\]

• required **memory** depends on the **number of accumulators** maintained

• **top-k results** can be determined by **sorting accumulators** at the end
1. Term-at-a-Time Query Processing

- **Term-at-a-Time** (TAAT) query processing
  - reads posting lists for query terms \( \langle t_1, \ldots, t_{|q|} \rangle \) successively
  - maintains an **accumulator** for each result document with value

\[
acc(d) = \sum_{i \leq j} score(t_i, d)
\]

after the first \( j \) posting lists have been read

\[
\begin{align*}
  a & \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad d_1, 1.0 \quad d_4, 2.0 \quad d_7, 0.2 \quad d_8, 0.1 \\
  b & \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad d_4, 1.0 \quad \textbf{d_7, 2.0} \quad d_8, 0.2 \quad d_9, 0.1 \\
  c & \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad d_4, 3.0 \quad d_7, 1.0
\end{align*}
\]

- **required memory** depends on the **number of accumulators** maintained

- **top-\( k \) results** can be determined by **sorting accumulators** at the end
1. Term-at-a-Time Query Processing

- **Term-at-a-Time** (TAAT) query processing
  - reads posting lists for query terms \( \langle t_1, \ldots, t_{|q|} \rangle \) **successively**
  - maintains an **accumulator** for each result document with value

\[
acc(d) = \sum_{i \leq j} score(t_i, d)
\]

after the first \( j \) posting lists have been read

- **required memory** depends on the **number of accumulators** maintained

- **top-\( k \) results** can be determined by **sorting accumulators** at the end

Accumulators

<table>
<thead>
<tr>
<th></th>
<th>( d_1 )</th>
<th>( d_4 )</th>
<th>( d_7 )</th>
<th>( d_8 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a )</td>
<td>( d_1 ), 1.0</td>
<td>( d_4 ), 2.0</td>
<td>( d_7 ), 0.2</td>
<td>( d_8 ), 0.1</td>
</tr>
<tr>
<td>( b )</td>
<td>( d_4 ), 1.0</td>
<td><strong>( d_7 ), 2.0</strong></td>
<td>( d_8 ), 0.2</td>
<td>( d_9 ), 0.1</td>
</tr>
<tr>
<td>( c )</td>
<td>( d_4 ), 3.0</td>
<td>( d_7 ), 1.0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
1. Term-at-a-Time Query Processing

- **Term-at-a-Time** (TAAT) query processing
  - reads posting lists for query terms \( t_1, \ldots, t_{|q|} \) **successively**
  - maintains an **accumulator** for each result document with value

\[
acc(d) = \sum_{i \leq j} score(t_i, d) \quad \text{after the first } j \text{ posting lists have been read}
\]

- **required memory** depends on the **number of accumulators** maintained

- **top-\( k \) results** can be determined by **sorting accumulators** at the end
1. Term-at-a-Time Query Processing

- **Term-at-a-Time** (TAAT) query processing
  - reads posting lists for query terms \( \langle t_1, \ldots, t_{|q|} \rangle \) **successively**
  - maintains an **accumulator** for each result document with value

\[
acc(d) = \sum_{i \leq j} score(t_i, d) \text{ after the first } j \text{ posting lists have been read}
\]

Accumulators

\[
\begin{array}{c|c|c|c|c|c}
\hline
\text{Doc} & d_1 & d_4 & d_7 & d_8 & d_9 \\
\hline
a & d_1, 1.0 & d_4, 2.0 & d_7, 0.2 & d_8, 0.1 & \\
\hline
b & d_4, 1.0 & d_7, 2.0 & \textbf{d}_8, 0.2 & d_9, 0.1 & \\
\hline
c & d_4, 3.0 & d_7, 1.0 & & & \\
\hline
\end{array}
\]

- required **memory** depends on the number of accumulators maintained

- **top-k results** can be determined by **sorting accumulators** at the end
1. Term-at-a-Time Query Processing

- **Term-at-a-Time (TAAT) query processing**
  - reads posting lists for query terms \( t_1, \ldots, t_{|q|} \) **successively**
  - maintains an **accumulator** for each result document with value

\[
acc(d) = \sum_{i \leq j} score(t_i, d) \quad \text{after the first} \; j \; \text{posting lists have been read}
\]

<table>
<thead>
<tr>
<th>Accumulators</th>
</tr>
</thead>
<tbody>
<tr>
<td>( d_1 ) : 1.0</td>
</tr>
<tr>
<td>( d_4 ) : 3.0</td>
</tr>
<tr>
<td>( d_7 ) : 2.2</td>
</tr>
<tr>
<td>( d_8 ) : 0.3</td>
</tr>
<tr>
<td>( d_9 ) : 0.0</td>
</tr>
</tbody>
</table>

- **required memory** depends on the **number of accumulators** maintained
- **top-\( k \) results** can be determined by **sorting accumulators** at the end
1. Term-at-a-Time Query Processing

- **Term-at-a-Time** (TAAT) query processing
  - reads posting lists for query terms \( \langle t_1, \ldots, t_{|q|} \rangle \) *successively*
  - maintains an **accumulator** for each result document with value

\[
\text{acc}(d) = \sum_{i \leq j} \text{score}(t_i, d) \quad \text{after the first } j \text{ posting lists have been read}
\]

<table>
<thead>
<tr>
<th>Document</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>(d_1), 1.0</td>
</tr>
<tr>
<td>b</td>
<td>(d_4), 1.0</td>
</tr>
<tr>
<td>c</td>
<td>(d_4), 3.0</td>
</tr>
</tbody>
</table>

- required **memory** depends on the **number of accumulators** maintained
- **top-k results** can be determined by **sorting accumulators** at the end
1. Term-at-a-Time Query Processing

- Term-at-a-Time (TAAT) query processing
  - reads posting lists for query terms \( \langle t_1, \ldots, t_{|q|} \rangle \) **successively**
  - maintains an **accumulator** for each result document with value

\[
acc(d) = \sum_{i \leq j} score(t_i, d) \quad \text{after the first} \ j \ \text{posting lists have been read}
\]

\[
\begin{array}{c|c|c|c|c}
   a & d_1, 1.0 & d_4, 2.0 & d_7, 0.2 & d_8, 0.1 \\
   b & d_4, 1.0 & d_7, 2.0 & d_8, 0.2 & d_9, 0.1 \\
   c & d_4, 3.0 & d_7, 1.0 & & \\
\end{array}
\]

- required **memory** depends on the number of accumulators maintained
- **top-k results** can be determined by **sorting accumulators** at the end
1. Term-at-a-Time Query Processing

- **Term-at-a-Time (TAAT)** query processing
  - reads posting lists for query terms $\langle t_1, \ldots, t_{|q|} \rangle$ **successively**
  - maintains an **accumulator** for each result document with value

$$
acc(d) = \sum_{i \leq j} score(t_i, d) \text{ after the first } j \text{ posting lists have been read}
$$

<table>
<thead>
<tr>
<th>$a$</th>
<th>$d_1$, 1.0</th>
<th>$d_4$, 2.0</th>
<th>$d_7$, 0.2</th>
<th>$d_8$, 0.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$b$</td>
<td>$d_4$, 1.0</td>
<td>$d_7$, 2.0</td>
<td>$d_8$, 0.2</td>
<td>$d_9$, 0.1</td>
</tr>
<tr>
<td>$c$</td>
<td>$d_4$, 3.0</td>
<td>$d_7$, 1.0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- required **memory** depends on the number of accumulators maintained
- **top-$k$ results** can be determined by **sorting accumulators** at the end
1. Term-at-a-Time Query Processing

- **Term-at-a-Time** (TAAT) query processing
  - reads posting lists for query terms $\langle t_1, \ldots, t_{|q|} \rangle$ **successively**
  - maintains an **accumulator** for each result document with value

$$acc(d) = \sum_{i \leq j} score(t_i, d)$$ after the first $j$ posting lists have been read

\[
\begin{array}{c|c|c|c|c|c|c|c}
\text{Accumulators} & d_1 & d_4 & d_7 & d_8 & d_9 \\
\hline
a & d_1, 1.0 & d_4, 2.0 & d_7, 0.2 & d_8, 0.1 \\
b & d_4, 1.0 & d_7, 2.0 & d_8, 0.2 & d_9, 0.1 \\
c & d_4, 3.0 & d_7, 1.0 \\
\end{array}
\]

- required **memory** depends on the **number of accumulators** maintained

- **top-$k$ results** can be determined by **sorting accumulators** at the end
1. Term-at-a-Time Query Processing

- **Term-at-a-Time (TAAT)** query processing
  - reads posting lists for query terms \( \langle t_1, \ldots, t_{|q|} \rangle \) **successively**
  - maintains an **accumulator** for each result document with value

\[
acc(d) = \sum_{i \leq j} score(t_i, d)
\]

after the first \( j \) posting lists have been read

\[
\begin{array}{l}
\text{Accumulators} \\
\hline
d_1 & : & 1.0 \\
d_4 & : & 6.0 \\
d_7 & : & 3.2 \\
d_8 & : & 0.3 \\
d_9 & : & 0.1 \\
\end{array}
\]

- required **memory** depends on the **number of accumulators** maintained
- **top-\( k \) results** can be determined by **sorting accumulators** at the end
1. Term-at-a-Time Query Processing

- **Term-at-a-Time** (TAAT) query processing
- reads posting lists for query terms \( \langle t_1, \ldots, t_{|q|} \rangle \) successively
- maintains an **accumulator** for each result document with value

\[
acc(d) = \sum_{i \leq j} score(t_i, d)
\]

after the first \( j \) posting lists have been read

<table>
<thead>
<tr>
<th>Accumulators</th>
</tr>
</thead>
<tbody>
<tr>
<td>( d_1 ) : 1.0</td>
</tr>
<tr>
<td>( d_4 ) : 6.0</td>
</tr>
<tr>
<td>( d_7 ) : 3.2</td>
</tr>
<tr>
<td>( d_8 ) : 0.3</td>
</tr>
<tr>
<td>( d_9 ) : 0.1</td>
</tr>
</tbody>
</table>

- required **memory** depends on the number of **accumulators** maintained
- **top-\( k \)** results can be determined by **sorting accumulators** at the end
Term-at-a-Time Query Processing

• Optimizations for **conjunctive queries**
  
  • process query terms in **ascending order of their document frequency** to keep the number of accumulators and thus required memory low
  
  • for document-ordered posting lists, **keep accumulators sorted** to make use of **skip pointers** when read posting lists
2. Document-at-a-Time Query Processing

• **Document-at-a-Time (DAAT)** query processing
  
  • assumes **document-ordered posting lists**
  
  • reads posting lists for query terms \( \langle t_1, \ldots, t_{|q|} \rangle \) **concurrently**
  
  • computes score when **same document** is seen in one or more posting lists

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(d_1, 1.0)</td>
<td>(d_4, 2.0)</td>
<td>(d_7, 0.2)</td>
<td>(d_8, 0.1)</td>
</tr>
<tr>
<td>b</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(d_4, 1.0)</td>
<td>(d_7, 2.0)</td>
<td>(d_8, 0.2)</td>
<td>(d_9, 0.1)</td>
</tr>
<tr>
<td>c</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(d_4, 3.0)</td>
<td>(d_7, 1.0)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

• always advances posting list with **lowest current document identifier**

• required main memory depends on the **number of results** to be reported

• **top-\( k \) results** can be determined by keeping results in **priority queue**
2. Document-at-a-Time Query Processing

- **Document-at-a-Time (DAAT)** query processing
  - assumes **document-ordered posting lists**
  - reads posting lists for query terms \( \langle t_1, \ldots, t_{|q|} \rangle \) **concurrently**
  - computes score when **same document** is seen in one or more posting lists

\[
\begin{align*}
\text{a} &\quad d_1, 1.0 & d_4, 2.0 & d_7, 0.2 & d_8, 0.1 \\
\text{b} &\quad d_4, 1.0 & d_7, 2.0 & d_8, 0.2 & d_9, 0.1 \\
\text{c} &\quad d_4, 3.0 & d_7, 1.0
\end{align*}
\]

- always advances posting list with **lowest current document identifier**
- required main memory depends on the **number of results** to be reported
- **top-\(k\) results** can be determined by keeping results in **priority queue**
2. Document-at-a-Time Query Processing

- **Document-at-a-Time (DAAT) query processing**
  - assumes document-ordered posting lists
  - reads posting lists for query terms \( \langle t_1, \ldots, t_{|q|} \rangle \) **concurrently**
  - computes score when **same document** is seen in one or more posting lists

\[
\begin{align*}
a &: d_1, 1.0 & d_4, 2.0 & d_7, 0.2 & d_8, 0.1 \\
b &: d_4, 1.0 & d_7, 2.0 & d_8, 0.2 & d_9, 0.1 \\
c &: d_4, 3.0 & d_7, 1.0
\end{align*}
\]

- always advances posting list with **lowest current document identifier**
- required main memory depends on the **number of results** to be reported
- **top-\(k\) results** can be determined by keeping results in **priority queue**
2. Document-at-a-Time Query Processing

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

  • assumes **document-ordered posting lists**

  • reads posting lists for query terms \( \langle t_1, \ldots, t_{|q|} \rangle \) **concurrently**

  • computes score when **same document** is seen in one or more posting lists

\[
\begin{align*}
\text{a} & : d_1, 1.0 \quad d_4, 2.0 \quad d_7, 0.2 \quad d_8, 0.1 \\
\text{b} & : d_4, 1.0 \quad d_7, 2.0 \quad d_8, 0.2 \quad d_9, 0.1 \\
\text{c} & : d_4, 3.0 \quad d_7, 1.0 \\
\end{align*}
\]

• always advances posting list with **lowest current document identifier**

• required main memory depends on the **number of results** to be reported

• **top-\(k\) results** can be determined by keeping results in **priority queue**
2. Document-at-a-Time Query Processing

- **Document-at-a-Time (DAAT)** query processing
  - assumes **document-ordered posting lists**
  - reads posting lists for query terms $\langle t_1, \ldots, t_{|q|} \rangle$ **concurrently**
  - computes score when **same document** is seen in one or more posting lists

<table>
<thead>
<tr>
<th>Document</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>$d_1$</td>
<td>1.0</td>
</tr>
<tr>
<td>$d_4$</td>
<td>6.0</td>
</tr>
</tbody>
</table>

- always advances posting list with **lowest current document identifier**
- required main memory depends on the **number of results** to be reported
- **top-k results** can be determined by keeping results in **priority queue**
2. Document-at-a-Time Query Processing

- **Document-at-a-Time (DAAT)** query processing
- assumes **document-ordered posting lists**
- reads posting lists for query terms \( \langle t_1, ..., t_{|q|} \rangle \) **concurrently**
- computes score when **same document** is seen in one or more posting lists

\[
\begin{align*}
\text{a} & \quad d_1, 1.0 \quad d_4, 2.0 \quad d_7, 0.2 \quad d_8, 0.1 \\
\text{b} & \quad d_4, 1.0 \quad d_7, 2.0 \quad d_8, 0.2 \quad d_9, 0.1 \\
\text{c} & \quad d_4, 3.0 \quad d_7, 1.0 \\
\end{align*}
\]

\[
\begin{align*}
\text{d}_1 & : 1.0 \\
\text{d}_4 & : 6.0
\end{align*}
\]

- always advances posting list with **lowest current document identifier**
- required main memory depends on the **number of results** to be reported
- **top-\(k\) results** can be determined by keeping results in **priority queue**
2. Document-at-a-Time Query Processing

- **Document-at-a-Time** (DAAT) query processing
  - assumes **document-ordered posting lists**
  - reads posting lists for query terms \( \langle t_1, \ldots, t_{|q|} \rangle \) **concurrently**
  - computes score when **same document** is seen in one or more posting lists

\[
\begin{align*}
  a & \quad d_1, 1.0 \quad d_4, 2.0 \quad d_7, 0.2 \quad d_8, 0.1 \\
  b & \quad d_4, 1.0 \quad d_7, 2.0 \quad d_8, 0.2 \quad d_9, 0.1 \\
  c & \quad d_4, 3.0 \quad d_7, 1.0 \\
\end{align*}
\]

- always advances posting list with **lowest current document identifier**
- required main memory depends on the **number of results** to be reported
- **top-k results** can be determined by keeping results in **priority queue**
2. Document-at-a-Time Query Processing

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

  - assumes **document-ordered posting lists**
  
  - reads posting lists for query terms \( \langle t_1, \ldots, t_{|q|} \rangle \) **concurrently**
  
  - computes score when **same document** is seen in one or more posting lists

- \( a \) ........... \( d_1, 1.0 \quad d_4, 2.0 \quad d_7, 0.2 \quad d_8, 0.1 \)

- \( b \) ........... \( d_4, 1.0 \quad d_7, 2.0 \quad d_8, 0.2 \quad d_9, 0.1 \)

- \( c \) ........... \( d_4, 3.0 \quad d_7, 1.0 \)

- always advances posting list with **lowest current document identifier**

- required main memory depends on the **number of results** to be reported

- **top-\( k \) results** can be determined by keeping results in **priority queue**
2. Document-at-a-Time Query Processing

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

  • assumes **document-ordered posting lists**
  
  • reads posting lists for query terms \( \langle t_1, \ldots, t_{|q|} \rangle \) **concurrently**
  
  • computes score when **same document** is seen in one or more posting lists

<table>
<thead>
<tr>
<th>a</th>
<th>( d_1, 1.0 )</th>
<th>( d_4, 2.0 )</th>
<th>( d_7, 0.2 )</th>
<th>( d_8, 0.1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>( d_4, 1.0 )</td>
<td><strong>( d_7, 2.0 )</strong></td>
<td>( d_8, 0.2 )</td>
<td>( d_9, 0.1 )</td>
</tr>
<tr>
<td>c</td>
<td>( d_4, 3.0 )</td>
<td>( d_7, 1.0 )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>d_1</th>
<th>1.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>d_4</td>
<td>6.0</td>
</tr>
<tr>
<td>d_7</td>
<td>3.2</td>
</tr>
</tbody>
</table>

• always advances posting list with **lowest current document identifier**

• required main memory depends on the **number of results** to be reported

• **top-\( k \) results** can be determined by keeping results in **priority queue**
2. Document-at-a-Time Query Processing

- **Document-at-a-Time** (DAAT) query processing
  - assumes document-ordered posting lists
  - reads posting lists for query terms \( \langle t_1, \ldots, t_{|q|} \rangle \) **concurrently**
  - computes score when **same document** is seen in one or more posting lists

- always advances posting list with **lowest current document identifier**
- required main memory depends on the **number of results** to be reported
- **top-k results** can be determined by keeping results in **priority queue**

\[
\begin{align*}
\text{a} & \quad d_1, 1.0 \quad d_4, 2.0 \quad d_7, 0.2 \quad d_8, 0.1 \\
\text{b} & \quad d_4, 1.0 \quad \text{**d_7, 2.0**} \quad d_8, 0.2 \quad d_9, 0.1 \\
\text{c} & \quad d_4, 3.0 \quad d_7, 1.0
\end{align*}
\]
2. Document-at-a-Time Query Processing

- **Document-at-a-Time (DAAT) query processing**
  - assumes **document-ordered posting lists**
  - reads posting lists for query terms \( \langle t_1, \ldots, t_{|q|} \rangle \) **concurrently**
  - computes score when **same document** is seen in one or more posting lists

    \[
    \begin{align*}
    a \quad & \quad d_1, 1.0 \quad d_4, 2.0 \quad d_7, 0.2 \quad d_8, 0.1 \\
    b \quad & \quad d_4, 1.0 \quad d_7, 2.0 \quad d_8, 0.2 \quad d_9, 0.1 \\
    c \quad & \quad d_4, 3.0 \quad d_7, 1.0
    \end{align*}
    \]

- always advances posting list with **lowest current document identifier**
- required main memory depends on the **number of results** to be reported
- **top-\( k \)** results can be determined by keeping results in **priority queue**

\[
\begin{align*}
   d_1 & : 1.0 \\
   d_4 & : 6.0 \\
   d_7 & : 3.2
\end{align*}
\]
2. Document-at-a-Time Query Processing

- **Document-at-a-Time (DAAT) query processing**
- assumes document-ordered posting lists
- reads posting lists for query terms \( \langle t_1, \ldots, t_{|q|} \rangle \) **concurrently**
- computes score when **same document** is seen in one or more posting lists

\[
\begin{align*}
a & \quad d_1, 1.0 \quad d_4, 2.0 \quad d_7, 0.2 \quad d_8, 0.1 \\
b & \quad d_4, 1.0 \quad d_7, 2.0 \quad d_8, 0.2 \quad d_9, 0.1 \\
c & \quad d_4, 3.0 \quad d_7, 1.0
\end{align*}
\]

- always advances posting list with **lowest current document identifier**
- required main memory depends on the **number of results** to be reported
- **top-\(k\) results** can be determined by keeping results in **priority queue**
2. Document-at-a-Time Query Processing

- **Document-at-a-Time (DAAT) query processing**
  - assumes **document-ordered posting lists**
  - reads posting lists for query terms $\langle t_1, \ldots, t_{|q|} \rangle$ **concurrently**
  - computes score when **same document** is seen in one or more posting lists

```
<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>$d_1$, 1.0</td>
<td>$d_4$, 2.0</td>
<td>$d_7$, 0.2</td>
</tr>
<tr>
<td>b</td>
<td>$d_4$, 1.0</td>
<td>$d_7$, 2.0</td>
<td>$d_8$, 0.2</td>
</tr>
<tr>
<td>c</td>
<td>$d_4$, 3.0</td>
<td>$d_7$, 1.0</td>
<td></td>
</tr>
</tbody>
</table>
```

- always advances posting list with **lowest current document identifier**
- required main memory depends on the **number of results** to be reported
- **top-$k$ results** can be determined by keeping results in **priority queue**
2. Document-at-a-Time Query Processing

- **Document-at-a-Time** (DAAT) query processing
  - assumes **document-ordered posting lists**
  - reads posting lists for query terms \( \langle t_1, \ldots, t_{|q|} \rangle \) **concurrently**
  - computes score when **same document** is seen in one or more posting lists

\[
\begin{align*}
a & \quad \ldots \quad d_1, 1.0 \quad d_4, 2.0 \quad d_7, 0.2 \quad d_8, 0.1 \\
b & \quad \ldots \quad d_4, 1.0 \quad d_7, 2.0 \quad d_8, 0.2 \quad \textcolor{red}{d_9, 0.1} \\
c & \quad \ldots \quad d_4, 3.0 \quad d_7, 1.0 \\
\end{align*}
\]

- always advances posting list with **lowest current document identifier**
- required main memory depends on the **number of results** to be reported
- **top-\(k\) results** can be determined by keeping results in **priority queue**
2. Document-at-a-Time Query Processing

- **Document-at-a-Time (DAAT) query processing**
  - assumes document-ordered posting lists
  - reads posting lists for query terms \( \langle t_1, \ldots, t_{|q|} \rangle \) **concurrently**
  - computes score when *same document* is seen in one or more posting lists

\[
\begin{align*}
a & \quad d_1, 1.0 \quad d_4, 2.0 \quad d_7, 0.2 \quad d_8, 0.1 \\
b & \quad d_4, 1.0 \quad d_7, 2.0 \quad d_8, 0.2 \quad \color{red}d_9, 0.1 \\
c & \quad d_4, 3.0 \quad d_7, 1.0
\end{align*}
\]

- always advances posting list with **lowest current document identifier**
- required main memory depends on the **number of results** to be reported
- **top-\(k\) results** can be determined by keeping results in **priority queue**
2. Document-at-a-Time Query Processing

• **Document-at-a-Time (DAAT) query processing**
  
  • assumes **document-ordered posting lists**
  
  • reads posting lists for query terms \( \langle t_1, \ldots, t_{|q|} \rangle \) **concurrently**
  
  • computes score when **same document** is seen in one or more posting lists

\[
\begin{align*}
  a & \quad d_1, 1.0 & d_4, 2.0 & d_7, 0.2 & d_8, 0.1 \\
  b & \quad d_4, 1.0 & d_7, 2.0 & d_8, 0.2 & d_9, 0.1 \\
  c & \quad d_4, 3.0 & d_7, 1.0 \\
\end{align*}
\]

• always advances posting list with **lowest current document identifier**

• required main memory depends on the **number of results** to be reported

• **top-\(k\) results** can be determined by keeping results in **priority queue**
Document-at-a-Time Query Processing

• Optimization for **conjunctive queries** using **skip pointers**

• when advancing posting list with lowest current document identifier, advance to first posting having document identifier **larger or equal to**

\[ \max_i \text{cdid}(i) \]

where \( \text{cdid}(i) \) is the current document identifier in the \( i \)-th posting list

---

<table>
<thead>
<tr>
<th>a</th>
<th>( d_1, 1.0 )</th>
<th>( d_4, 2.0 )</th>
<th>( d_7, 0.2 )</th>
<th>( d_8, 0.1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>( d_4, 1.0 )</td>
<td>( d_7, 2.0 )</td>
<td>( d_8, 0.2 )</td>
<td>( d_9, 0.1 )</td>
</tr>
<tr>
<td>c</td>
<td>( d_4, 3.0 )</td>
<td>( d_7, 1.0 )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Document-at-a-Time Query Processing

- Optimization for **conjunctive queries** using **skip pointers**

  - when advancing posting list with lowest current document identifier, advance to first posting having document identifier **larger or equal to**

    \[
    \max_i cdid(i)
    \]

    where \( cdid(i) \) is the current document identifier in the \( i \)-th posting list

<table>
<thead>
<tr>
<th></th>
<th>d1, 1.0</th>
<th>d4, 2.0</th>
<th>d7, 0.2</th>
<th>d8, 0.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>d4, 1.0</td>
<td>d7, 2.0</td>
<td>d8, 0.2</td>
<td>d9, 0.1</td>
</tr>
<tr>
<td>c</td>
<td>d4, 3.0</td>
<td>d7, 1.0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Document-at-a-Time Query Processing

• Optimization for conjugate queries using skip pointers

• when advancing posting list with lowest current document identifier, advance to first posting having document identifier larger or equal to

\[
\max_i \text{cdid}(i)
\]

where \( \text{cdid}(i) \) is the current document identifier in the \( i \)-th posting list

\[
\begin{align*}
a & \quad d_1, 1.0 & d_4, 2.0 & d_7, 0.2 & d_8, 0.1 \\ b & \quad d_4, 1.0 & d_7, 2.0 & d_8, 0.2 & d_9, 0.1 \\ c & \quad d_4, 3.0 & d_7, 1.0 \\
\end{align*}
\]

\[d_4 : 6.0\]
Document-at-a-Time Query Processing

- Optimization for **conjunctive queries using skip pointers**

  - when advancing posting list with lowest current document identifier, advance to first posting having document identifier **larger or equal to**

    $$\max_i \text{cdid}(i)$$

  where \( \text{cdid}(i) \) is the current document identifier in the \( i \)-th posting list

\[
\begin{align*}
  a & : d_1, 1.0 & d_4, 2.0 & d_7, 0.2 & d_8, 0.1 \\
  b & : d_4, 1.0 & d_7, 2.0 & d_8, 0.2 & d_9, 0.1 \\
  c & : d_4, 3.0 & d_7, 1.0 \\
\end{align*}
\]
Document-at-a-Time Query Processing

- Optimization for **conjunctive queries using skip pointers**

- when advancing posting list with lowest current document identifier, advance to first posting having document identifier **larger or equal to**

\[
\max_i \text{cdid}(i)
\]

where \(\text{cdid}(i)\) is the current document identifier in the \(i\)-th posting list

<table>
<thead>
<tr>
<th></th>
<th>(d_1), 1.0</th>
<th>(d_4), 2.0</th>
<th>(d_7), 0.2</th>
<th>(d_8), 0.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a)</td>
<td>(d_4), 1.0</td>
<td>(d_7), 2.0</td>
<td>(d_8), 0.2</td>
<td>(d_9), 0.1</td>
</tr>
</tbody>
</table>

\[d_4 : 6.0\]
Document-at-a-Time Query Processing

• Optimization for **conjunctive queries using skip pointers**

• when advancing posting list with lowest current document identifier, advance to first posting having document identifier **larger or equal to**

\[
\max_i \text{cdid}(i)
\]

where cdid(i) is the current document identifier in the i-th posting list

\[
\begin{array}{c|c|c|c|c}
\text{a} & d_1, 1.0 & d_4, 2.0 & d_7, 0.2 & d_8, 0.1 \\
\text{b} & d_4, 1.0 & d_7, 2.0 & d_8, 0.2 & d_9, 0.1 \\
\text{c} & d_4, 3.0 & d_7, 1.0 & & \\
\end{array}
\]

\[
d_4 : 6.0
\]
Document-at-a-Time Query Processing

- Optimization for **conjunctive queries** using **skip pointers**

  - when advancing posting list with lowest current document identifier, advance to first posting having document identifier **larger or equal to**

    \[
    \max_i \text{cdid}(i)
    \]

  where \(\text{cdid}(i)\) is the current document identifier in the \(i\)-th posting list

\[
\begin{array}{c|c|c|c|c}
 a & \text{...........} & d_1, 1.0 & d_4, 2.0 & d_7, 0.2 & d_8, 0.1 \\
 b & \text{...........} & d_4, 1.0 & d_7, 2.0 & d_8, 0.2 & d_9, 0.1 \\
 c & \text{...........} & d_4, 3.0 & \textcolor{red}{d_7}, 1.0 \\
\end{array}
\]

\[
\begin{array}{c|c}
 d_4 & 6.0 \\
 d_7 & 3.2 \\
\end{array}
\]
Document-at-a-Time Query Processing

• Optimization for **conjunctive queries** using **skip pointers**

• when advancing posting list with lowest current document identifier, advance to first posting having document identifier **larger or equal to** 

\[
\max_i \text{cdid}(i)
\]

where \(\text{cdid}(i)\) is the current document identifier in the \(i\)-th posting list

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(a)</td>
<td>(d_1, 1.0)</td>
<td>(d_4, 2.0)</td>
<td>(d_7, 0.2)</td>
</tr>
<tr>
<td>(b)</td>
<td>(d_4, 1.0)</td>
<td>(\textbf{d_7, 2.0})</td>
<td>(d_8, 0.2)</td>
</tr>
<tr>
<td>(c)</td>
<td>(d_4, 3.0)</td>
<td>(\textbf{d_7, 1.0})</td>
<td></td>
</tr>
</tbody>
</table>

\[
\begin{align*}
\text{d}_4 & : 6.0 \\
\text{d}_7 & : 3.2
\end{align*}
\]
Document-at-a-Time Query Processing

- Optimization for **conjunctive queries** using **skip pointers**

- when advancing posting list with lowest current document identifier, advance to first posting having document identifier **larger or equal to**

\[ \max_i \text{cdid}(i) \]

where \( \text{cdid}(i) \) is the current document identifier in the \( i \)-th posting list

\[ \begin{align*}
    a & \quad \text{........} \quad d_1, 1.0 \quad d_4, 2.0 \quad d_7, 0.2 \quad d_8, 0.1 \\
    b & \quad \text{........} \quad d_4, 1.0 \quad \textbf{d_7, 2.0} \quad d_8, 0.2 \quad d_9, 0.1 \\
    c & \quad \text{........} \quad d_4, 3.0 \quad d_7, 1.0 \\
\end{align*} \]

\[ \begin{align*}
    d_4 & : \quad 6.0 \\
    d_7 & : \quad 3.2
\end{align*} \]
Document-at-a-Time Query Processing

• Optimization for **conjunctive queries** using **skip pointers**

• when advancing posting list with lowest current document identifier, advance to first posting having document identifier **larger or equal to**

\[
\max_i \text{cdid}(i)
\]

where \( \text{cdid}(i) \) is the current document identifier in the \( i \)-th posting list

<table>
<thead>
<tr>
<th>a</th>
<th>d_1, 1.0</th>
<th>d_4, 2.0</th>
<th>d_7, 0.2</th>
<th>d_8, 0.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>d_4, 1.0</td>
<td>d_7, 2.0</td>
<td>d_8, 0.2</td>
<td>d_9, 0.1</td>
</tr>
<tr>
<td>c</td>
<td>d_4, 3.0</td>
<td>d_7, 1.0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[
\begin{align*}
\text{d}_4 & : 6.0 \\
\text{d}_7 & : 3.2
\end{align*}
\]
3. WAND

• **Weak AND** (WAND) query processing
  
  • assumes **document-ordered posting lists** with known maximum score \( \text{maxscore}(i) \) of any posting in the \( i \)-th posting list
  
  • reads posting lists for query terms \( \langle t_1, \ldots, t_{|q|} \rangle \) **concurrently**
  
  • computes score when **same document** is seen in one or more posting lists
  
  • always advances posting list with **lowest current document identifier** up to **pivot document identifier** computed from **current top-\( k \) result**

- **Computation of pivot document identifier**

  • let \( \min_k \) denote the **lowest score** in current top-\( k \) results
  
  • sort posting lists in **ascending order** of \( \text{cdid}(i) \)
  
  • pivot is \( \text{cdid}(j) \) of **minimal** \( j \) such that \( \min_k < \sum_{i \leq j} \text{maxscore}(i) \)
• Computation of **pivot document identifier**

  • let $min_k$ denote the **lowest score** in current top-$k$ results

  • sort posting lists in **ascending order** of $cdid(i)$

  • pivot is $cdid(j)$ of minimal $j$ such that $min_k < \sum_{i \leq j} maxscore(i)$

<table>
<thead>
<tr>
<th>a</th>
<th>d2, 0.5</th>
<th>d7, 0.1</th>
<th>d8, 0.2</th>
<th>d9, 0.6</th>
<th></th>
<th>d99, 1.0</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td></td>
<td>d2, 0.5</td>
<td>d9, 0.3</td>
<td>d11, 0.2</td>
<td>d13, 0.1</td>
<td></td>
<td>d33, 1.0</td>
</tr>
<tr>
<td>c</td>
<td></td>
<td>d2, 0.5</td>
<td>d3, 0.4</td>
<td>d4, 0.2</td>
<td>d5, 0.1</td>
<td></td>
<td>d57, 1.0</td>
</tr>
</tbody>
</table>

**Top-1**

$d_2 : 1.5$

<table>
<thead>
<tr>
<th>Pivot Computation</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>d3, 0.4</td>
<td>1.0</td>
</tr>
<tr>
<td>d7, 0.1</td>
<td>2.0</td>
</tr>
<tr>
<td>d9, 0.3</td>
<td>3.0</td>
</tr>
</tbody>
</table>

$maxscore(i) = 1.0$

$d_7$ is pivot
WAND

• **Intuition**: No document with an identifier smaller than the pivot can have a **score large enough** to make it into the top-\(k\) result

• **Observation**: As the value of \(\min_k\) **can only increase** over time, WAND **skips more and more postings** as time progresses

• WAND can be made an **approximate top-\(k\) query processing method** by computing the pivot such that

\[
F \times \min_k < \sum_{i \leq j} \maxscore(i)
\]

with **tunable parameter** \(F\) controlling fidelity of results

• **Full details**: [Broder et al. ’03]
4. Quit & Continue

• Quit & Continue query processing
  • reads score-ordered posting lists for query terms $\langle t_1, \ldots, t_{\|q\|} \rangle$ successively in descending order of $idf(t_i)$

• Quit heuristics
  • ignore posting lists for terms $t_i$ with $idf(t_i)$ below threshold
  • stop scanning posting list for $t_i$ if $tf(t_i, d_j)*idf(t_i)$ drops below threshold
  • stop scanning posting list when the number of accumulators is too high

• Continue heuristics
  • upon reaching accumulator limit, continue reading remaining posting lists, update existing accumulators but do not create new accumulators

• Full details: [Moffat and Zobel ’96]
5. Buckley’s Algorithm

- **Buckley’s** query processing method
  - reads **score-ordered posting lists concurrently** in round-robin manner
  - maintains **partial scores** of documents and keeps track of \( k \)-th best score
  - computes **upper bound** for any unseen document based on current scores
    \[
    ub = \sum_{i} cscore(i)
    \]
    with \( cscore(i) \) as the **current score** in the \( i \)-th posting list
  - **stops** if upper bound \( ub \) is less than \( k \)-th best partial score

\[
\begin{align*}
\text{Top-1} & \\
\begin{array}{|c|c|}
\hline
 & \\
\text{a} & \text{b} & \text{c} \\
\hline
\end{array}
\end{align*}
\]

\[
\begin{array}{|c|c|}
\hline
 & \text{d} & \text{e} & \text{f} & \text{g} & \text{h} & \text{i} & \text{j} \\
\hline
\text{d}_{2} & 0.5 & \text{d}_{1} & 0.4 & \text{d}_{5} & 0.3 & \text{d}_{9} & 0.2 \\
\hline
\text{d}_{2} & 0.5 & \text{d}_{3} & 0.5 & \text{d}_{6} & 0.4 & \text{d}_{13} & 0.1 \\
\hline
\text{d}_{3} & 0.4 & \text{d}_{5} & 0.3 & \text{d}_{7} & 0.2 & \text{d}_{4} & 0.1 \\
\hline
\end{array}
\]

\[
ub = 0.9
\]
Buckley’s Algorithm

• **Note**: This is a simplified version of Buckley’s algorithm. The original algorithm maintains an upper bound for the \((k + 1)\)-th best document. If implemented correctly, this gives us the **first exact top-\(k\) query processing method** described in the literature, which is only based on sequential accesses.

• **Full details**: [Buckley and Lewitt ’85]
6. Fagin’s Threshold Algorithms

• **Threshold Algorithm** (TA)
  - original version, often used as synonym for entire family of algorithms
  - requires eager random access to candidate objects
  - worst-case memory consumption: $O(k)$

• **No-Random-Accesses** (NRA)
  - no random access required, may have to scan large parts of the lists
  - worst-case memory consumption: $O(m*n + k)$

• **Combined Algorithm** (CA)
  - cost-model for scheduling random accesses to candidate objects
  - algorithmic skeleton very similar to NRA, but typically terminates faster
  - worst-case memory consumption: $O(m*n + k)$
Fagin’s Threshold Algorithms

- Assume **score-ordered posting lists** and **additional index** for score look-ups by document identifier

- Scan posting lists using **inexpensive sequential accesses** (SA) in round-robin manner

- Perform **expensive random accesses** (RA) to look up scores for a specific document when beneficial

- Support **monotone score aggregation function**

  \[
  aggr : \mathbb{R}^m \rightarrow \mathbb{R} : \forall x_i \geq x'_i \Rightarrow aggr(x_1, \ldots, x_m) \geq aggr(x'_1, \ldots, x'_m)
  \]

- Compute **aggregate scores** incrementally in **candidate queue**

- Compute **score bounds** for candidate results and stop when **threshold test** guarantees correct top-\(k\) result
Threshold Algorithm (TA)

- **Sequential accesses (SA)** mixed with eager random accesses (RA)
- Worst-case memory consumption $O(k)$

Threshold Algorithm (TA):
- scan index lists (e.g., round-robin)
- consider $d = cdid(i)$ in posting list for $t_i$
- $high(i) = cscore(i)$

```plaintext
if $d \notin top-k$ then // compute score$(d)$
    look up score$(t_j, d)$ for all $j \neq i$
    score$(d) = aggr\{score(t_j, d) \mid j = 1 \ldots |q|\}$

if score$(d) > \text{min}-k$ then // update top-$k$
    add $d$ to top-$k$ and remove min-score $d'$
    $\text{min}_k = \text{min}\{score(d') \mid d' \in top-k\}$

$\text{ub} = aggr\{high(i) \mid i = 1 \ldots |q|\}$ // update upper bound

if $\text{ub} \leq \text{min}_k$ then
    exit
```

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td></td>
<td>$d_{78}$, 0.9</td>
<td>$d_{23}$, 0.8</td>
<td>$d_{10}$, 0.8</td>
<td>$d_1$, 0.7</td>
<td>$d_{88}$, 0.2</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td></td>
<td>$d_{64}$, 0.9</td>
<td>$d_{23}$, 0.6</td>
<td>$d_{10}$, 0.6</td>
<td>$d_{12}$, 0.2</td>
<td>$d_{78}$, 0.1</td>
<td></td>
</tr>
<tr>
<td>c</td>
<td></td>
<td>$d_{10}$, 0.7</td>
<td>$d_{78}$, 0.5</td>
<td>$d_{64}$, 0.3</td>
<td>$d_{99}$, 0.2</td>
<td>$d_{34}$, 0.1</td>
<td></td>
</tr>
</tbody>
</table>

Top-2

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>SA</td>
<td>RA</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Threshold Algorithm (TA)

- Sequential accesses (SA) mixed with eager random accesses (RA)
- Worst-case memory consumption $O(k)$

Threshold Algorithm (TA):
scan index lists (e.g., round-robin)
consider $d = cdid(i)$ in posting list for $t_i$
$high(i) = cscore(i)$

\[
\text{if } d \notin \text{top-}k \text{ then} \quad \quad // \text{compute score}(d) \\
\text{look up score}(t_j, d) \text{ for all } j \neq i \\
\text{score}(d) = \text{aggr}\{ \text{score}(t_j, d) | j = 1 \ldots |q| \} \\
\]

\[
\text{if score}(d) > \text{min-}k \text{ then} \quad \quad // \text{update top-}k \\
\text{add } d \text{ to top-}k \text{ and remove min-score } d' \\
\text{min}_k = \text{min}\{ \text{score}(d') | d' \in \text{top-}k \} \\
\]

\[
\text{ub} = \text{aggr}\{ \text{high}(i) | i = 1 \ldots |q| \} \quad // \text{update upper bound} \\
\text{if ub} \leq \text{min}_k \text{ then} \\
\text{exit} \\
\]

\[
\begin{align*}
\text{a} & : \quad d_{78}, 0.9 \quad d_{23}, 0.8 \quad d_{10}, 0.8 \quad d_1, 0.7 \quad d_{88}, 0.2 \\
\text{b} & : \quad d_{64}, 0.9 \quad d_{23}, 0.6 \quad d_{10}, 0.6 \quad d_{12}, 0.2 \quad d_{78}, 0.1 \\
\text{c} & : \quad d_{10}, 0.7 \quad d_{78}, 0.5 \quad d_{64}, 0.3 \quad d_{99}, 0.2 \quad d_{34}, 0.1 \\
\text{ub} & = 2.5
\end{align*}
\]

Top-2
\[
\begin{align*}
d_{10} & : \quad 2.1 \\
d_{78} & : \quad 1.5
\end{align*}
\]
Threshold Algorithm (TA)

- **Sequential accesses (SA)** mixed with eager **random accesses (RA)**
- Worst-case memory consumption $O(k)$

Threshold Algorithm (TA):
scan index lists (e.g., round-robin)
consider $d = cdid(i)$ in posting list for $t_i$
$high(i) = cscore(i)$

if $d \notin \text{top-}k$ then  \hspace{0.3cm} // compute score($d$)
look up score($t_j, d$) for all $j \neq i$
score($d$) = aggr\{ score($t_j, d$) | $j = 1$ \ldots |q| \}

if score($d$) > min-$k$ then  \hspace{0.3cm} // update top-$k$
add $d$ to top-$k$ and remove min-score $d'$
min$_k = \min\{ \text{score}(d') | d' \in \text{top-}k \}$

$ub = \text{aggr}\{ high(i) | i = 1$ \ldots |q| \}$  \hspace{0.3cm} // update upper bound
if $ub \leq \text{min}_k$ then
exit

<table>
<thead>
<tr>
<th>a</th>
<th>$d_{78}, 0.9$</th>
<th>$d_{23}, 0.8$</th>
<th>$d_{10}, 0.8$</th>
<th>$d_{1}, 0.7$</th>
<th>$d_{88}, 0.2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>$d_{64}, 0.9$</td>
<td>$d_{23}, 0.6$</td>
<td>$d_{10}, 0.6$</td>
<td>$d_{12}, 0.2$</td>
<td>$d_{78}, 0.1$</td>
</tr>
<tr>
<td>c</td>
<td>$d_{10}, 0.7$</td>
<td>$d_{78}, 0.5$</td>
<td>$d_{64}, 0.3$</td>
<td>$d_{99}, 0.2$</td>
<td>$d_{34}, 0.1$</td>
</tr>
</tbody>
</table>

$ub = 1.9$

Top-2

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$d_{10}$</td>
<td>2.1</td>
<td></td>
</tr>
<tr>
<td>$d_{78}$</td>
<td>1.5</td>
<td></td>
</tr>
</tbody>
</table>

SA  RA
Threshold Algorithm (TA)

- **Sequential accesses (SA)**
  - mixed with eager **random accesses (RA)**

- **Worst-case memory consumption** \(O(k)\)

---

**Threshold Algorithm (TA):**

- scan index lists (e.g., round-robin)
- consider \(d = cdid(i)\) in posting list for \(t_i\)
- \(high(i) = cscore(i)\)

```plaintext
if d ∉ top-k then
  // compute score(d)
  look up score(t_j, d) for all j ≠ i
  score(d) = aggr{ score(t_j, d) | j = 1 ... |q| }

if score(d) > min_k then
  // update top-k
  add d to top-k and remove min-score \(d'\)
  \(min_k = min\{ score(d') | d' ∈ top-k \}\)

\(ub = aggr\{high(i) | i = 1 ... |q|\}\)  // update upper bound

if ub ≤ min_k then
  exit
```

---

**Top-2**

<p>| | | | | | | | | |</p>
<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>a</td>
<td></td>
<td></td>
<td>d_78, 0.9</td>
<td>d_23, 0.8</td>
<td>d_10, 0.8</td>
<td>d_1, 0.7</td>
<td>d_88, 0.2</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td></td>
<td></td>
<td>d_64, 0.9</td>
<td>d_23, 0.6</td>
<td>d_10, 0.6</td>
<td>d_12, 0.2</td>
<td>d_78, 0.1</td>
<td></td>
</tr>
<tr>
<td>c</td>
<td></td>
<td></td>
<td>d_10, 0.7</td>
<td>d_78, 0.5</td>
<td>d_64, 0.3</td>
<td>d_99, 0.2</td>
<td>d_34, 0.1</td>
<td></td>
</tr>
</tbody>
</table>

\(ub = 1.7\)
Threshold Algorithm (TA)

- **Sequential accesses (SA)**
  mixed with eager **random accesses (RA)**

- Worst-case memory consumption $O(k)$

---

**Threshold Algorithm (TA):**
scan index lists (e.g., round-robin)
consider $d = cdid(i)$ in posting list for $t_i$
$high(i) = cscore(i)$

```
if $d \not\in$ top-$k$ then // compute score($d$)
    look up score($t_j, d$) for all $j \neq i$
    score($d$) = aggr{ score($t_j, d$) | $j = 1 \ldots |q|$ }

if score($d$) $>$ min-$k$ then // update top-$k$
    add $d$ to top-$k$ and remove min-score $d'$
    min$_k$ = min{ score($d'$) | $d' \in$ top-$k$ }

ub = aggr{ high($i$) | $i = 1 \ldots |q|$ } // update upper bound
if ub $\leq$ min$_k$ then
    exit
```

**Top-2**

```
da_{10} : 2.1
da_{78} : 1.5
```

---

```
a ............ d_{78}, 0.9  d_{23}, 0.8  d_{10}, 0.8  d_{1}, 0.7  d_{88}, 0.2 ............
b ............ d_{64}, 0.9  d_{23}, 0.6  d_{10}, 0.6  d_{12}, 0.2  d_{78}, 0.1 ............
c ............ d_{10}, 0.7  d_{78}, 0.5  d_{64}, 0.3  d_{99}, 0.2  d_{34}, 0.1 ............
```

$ub = 1.1$

---

**Legend:**
- **SA**
- **RA**
Threshold Algorithm (TA)

- **Sequential accesses (SA)** mixed with eager **random accesses (RA)**
- Worst-case memory consumption $O(k)$

---

**Threshold Algorithm (TA):**
- scan index lists (e.g., round-robin)
- consider $d = cdid(i)$ in posting list for $t_i$
- $high(i) = cscore(i)$

```
if $d \notin top-k$ then  // compute $score(d)$
    look up $score(t_j, d)$ for all $j \neq i$
    $score(d) = aggr\{ score(t_j, d) | j = 1 \ldots |q| \}$

if $score(d) > min-k$ then  // update top-$k$
    add $d$ to top-$k$ and remove min-score $d'$
    $min_k = min\{ score(d') | d' \in top-k \}$

$ub = aggr\{ high(i) | i = 1 \ldots |q| \}$  // update upper bound
if $ub \leq min_k$ then
    exit
```

- $a$ ........... $d_{78}, 0.9$ $d_{23}, 0.8$ $d_{10}, 0.8$ $d_1, 0.7$ $d_{88}, 0.2$ ...........
- $b$ ........... $d_{64}, 0.9$ $d_{23}, 0.6$ $d_{10}, 0.6$ $d_{12}, 0.2$ $d_{78}, 0.1$ ...........
- $c$ ........... $d_{10}, 0.7$ $d_{78}, 0.5$ $d_{64}, 0.3$ $d_{99}, 0.2$ $d_{34}, 0.1$ ...........

$ub = 1.1$

---

Top-2

```
$\begin{array}{ll}
d_{10} & : 2.1 \\
d_{78} & : 1.5 \\
\end{array}$
```

**SA**  **RA**
No-Random-Accesses Algorithm (NRA)

- **Sequential accesses** (SA) only
- Worst-case memory consumption \(O(m*n + k)\)

No-Random-Accesses Algorithm (NRA):
scan index lists (e.g., round-robin)
consider \(d = cdid(i)\) in posting list for \(t_i\)
\(high(i) = cscore(i)\)
\(eval(d) = eval(d) \cup \{i\}\) // where have we seen \(d\)?
\(worst(d) = aggr\{ score(t_j, d) | j \in eval(d) \}\)
\(best(d) = aggr\{ worst(d), aggr\{ high(j) | j \notin eval(d) \} \}\)

\[
\text{if } worst(d) > min_k \text{ then} \quad \text{// good enough for top-k?}
\quad \text{add } d \text{ top top-k}
\quad min_k = \min\{ worst(d’) | d’ \in \text{top-k} \}
\]

\[
\text{else if } best(d) > min_k \text{ then} \quad \text{// good enough for cand?}
\quad \text{cand} = \text{cand} \cup \{ d \}
\quad ub = \max\{ best(d’) | d’ \in \text{cand} \}
\]

\[
\text{if } ub \leq min_k \text{ then}
\quad \text{exit}
\]

\[
\begin{array}{llllll}
\text{a} & \ldots & d_{78}, 0.9 & d_{23}, 0.8 & d_{10}, 0.8 & d_1, 0.7 & d_{88}, 0.2 & \ldots \\
\text{b} & \ldots & d_{64}, 0.8 & d_{23}, 0.6 & d_{10}, 0.6 & d_{12}, 0.2 & d_{78}, 0.1 & \ldots \\
\text{c} & \ldots & d_{10}, 0.7 & d_{78}, 0.5 & d_{64}, 0.3 & d_{99}, 0.2 & d_{34}, 0.1 & \ldots \\
\end{array}
\]
No-Random-Accesses Algorithm (NRA)

- **Sequential accesses (SA) only**
- **Worst-case memory consumption** \(O(m*n + k)\)

No-Random-Accesses Algorithm (NRA):
scan index lists (e.g., round-robin)
consider \(d = cid(i)\) in posting list for \(t_i\)
\(high(i) = cscore(i)\)
\(eval(d) = eval(d) \cup \{i\}\) // where have we seen \(d\)?

\begin{align*}
worst(d) &= \text{aggr}\{\text{score}(t_j, d) | j \in eval(d)\} \\
best(d) &= \text{aggr}\{\text{worst}(d), \text{aggr}\{\text{high}(j) | j \notin eval(d)\}\} \\
if \text{worst}(d) > min_k \text{ then} & \quad \text{// good enough for top-k?}\ 
\text{add } d \text{ top top-k}
min_k = \text{min}\{\text{worst}(d') | d' \in \text{top-k}\}
\text{else if } \text{best}(d) > min_k \text{ then} & \quad \text{// good enough for cand?}\ 
\text{cand} = \text{cand} \cup \{d\}
ub = \text{max}\{\text{best}(d') | d' \in \text{cand}\}
\text{if } ub \leq min_k \text{ then} & \quad \text{exit}
\end{align*}

\[
\begin{array}{ccc}
a & d_{78}, 0.9 & d_{23}, 0.8 & d_{10}, 0.8 & d_{1}, 0.7 & d_{88}, 0.2 & \ldots \\
b & d_{64}, 0.8 & d_{23}, 0.6 & d_{10}, 0.6 & d_{12}, 0.2 & d_{78}, 0.1 & \ldots \\
c & d_{10}, 0.7 & d_{78}, 0.5 & d_{64}, 0.3 & d_{99}, 0.2 & d_{34}, 0.1 & \ldots \\
\end{array}
\]

\[ub = 2.4\]

<table>
<thead>
<tr>
<th>\text{worst}</th>
<th>\text{best}</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{d}_{78} : 0.9 : 2.4</td>
<td></td>
</tr>
<tr>
<td>\text{d}_{64} : 0.8 : 2.4</td>
<td></td>
</tr>
<tr>
<td>\text{d}_{10} : 0.7 : 2.4</td>
<td></td>
</tr>
</tbody>
</table>

**Top-1**

- **SA**
- **RA**

IR&DM ’13/’14
No-Random-Accesses Algorithm (NRA)

- **Sequential accesses (SA) only**
- **Worst-case memory consumption** $O(m*n + k)$

No-Random-Accesses Algorithm (NRA):
scan index lists (e.g., round-robin)
consider $d = \text{cdid}(i)$ in posting list for $t_i$
$\text{high}(i) = \text{cscore}(i)$
$\text{eval}(d) = \text{eval}(d) \cup \{i\}$ // where have we seen $d$?

$\text{worst}(d) = \text{aggr}\{ \text{score}(t_j, d) | j \in \text{eval}(d) \}$
$\text{best}(d) = \text{aggr}\{ \text{worst}(d), \text{aggr}\{ \text{high}(j) | j \notin \text{eval}(d) \} \}$

**if** $\text{worst}(d) > \min_k$ **then** // good enough for top-$k$?
  add $d$ to $\text{top}$
  $\min_k = \min\{ \text{worst}(d') | d' \in \text{top}-k \}$
**else if** $\text{best}(d) > \min_k$ **then** // good enough for cand?
  $\text{cand} = \text{cand} \cup \{ d \}$
  $\text{ub} = \max\{ \text{best}(d') | d' \in \text{cand} \}$
**if** $\text{ub} \leq \min_k$ **then**
  exit

<table>
<thead>
<tr>
<th></th>
<th>$a$</th>
<th>$b$</th>
<th>$c$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$d_{78}$</td>
<td>0.9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$d_{23}$</td>
<td></td>
<td>0.6</td>
<td></td>
</tr>
<tr>
<td>$d_{10}$</td>
<td></td>
<td>0.8</td>
<td></td>
</tr>
<tr>
<td>$d_{64}$</td>
<td>0.8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$d_{12}$</td>
<td></td>
<td>0.2</td>
<td></td>
</tr>
<tr>
<td>$d_{1}$</td>
<td></td>
<td></td>
<td>0.6</td>
</tr>
<tr>
<td>$d_{34}$</td>
<td></td>
<td></td>
<td>0.1</td>
</tr>
<tr>
<td>$d_{99}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$d_{88}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$d_{78}$</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>worst</th>
<th>best</th>
</tr>
</thead>
<tbody>
<tr>
<td>$d_{78}$</td>
<td>1.4</td>
<td>2.0</td>
</tr>
<tr>
<td>$d_{23}$</td>
<td>1.4</td>
<td>1.9</td>
</tr>
<tr>
<td>$d_{64}$</td>
<td>0.8</td>
<td>2.1</td>
</tr>
<tr>
<td>$d_{10}$</td>
<td>0.7</td>
<td>2.1</td>
</tr>
</tbody>
</table>

$\text{ub} = 2.1$

**Top-$I$**

SA  RA
### No-Random-Accesses Algorithm (NRA)

- **Sequential accesses (SA) only**

- **Worst-case memory consumption** $O(m*n + k)$

---

#### No-Random-Accesses Algorithm (NRA):

- scan index lists (e.g., round-robin)
- consider $d = cdid(i)$ in posting list for $t_i$
- $high(i) = cscore(i)$
- $eval(d) = eval(d) \cup \{i\}$ // where have we seen $d$?

- $worst(d) = \text{aggr}\{\text{score}(t_j, d) | j \in eval(d)\}$
- $best(d) = \text{aggr}\{worst(d), \text{aggr}\{high(j) | j \notin eval(d)\}\}$

**Pseudo-code**

```
if worst(d) > min_k then  // good enough for top-k?
    add d top top-k
    min_k = min\{worst(d') | d' \in top-k\}
else if best(d) > min_k then  // good enough for cand?
    cand = cand \cup \{d\}
    ub = max\{best(d') | d' \in cand\}
if ub \leq min_k then  
    exit
```

---

#### Example

<table>
<thead>
<tr>
<th></th>
<th>worst</th>
<th>best</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>$d_{78}$, 0.9</td>
<td>$d_{10}$, 2.1</td>
</tr>
<tr>
<td></td>
<td>$d_{23}$, 0.8</td>
<td>$d_{1}$, 2.1</td>
</tr>
<tr>
<td></td>
<td>$d_{88}$, 0.2</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>worst</th>
<th>best</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>$d_{64}$, 0.8</td>
<td>$d_{10}$, 2.1</td>
</tr>
<tr>
<td></td>
<td>$d_{23}$, 0.6</td>
<td>$d_{12}$, 2.1</td>
</tr>
<tr>
<td></td>
<td>$d_{78}$, 0.1</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>worst</th>
<th>best</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>$d_{10}$, 0.7</td>
<td>$d_{88}$, 0.2</td>
</tr>
<tr>
<td></td>
<td>$d_{78}$, 0.5</td>
<td>$d_{99}$, 0.2</td>
</tr>
<tr>
<td></td>
<td>$d_{64}$, 0.3</td>
<td>$d_{34}$, 0.1</td>
</tr>
</tbody>
</table>

ub = 2.0

---

#### Top-1

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$d_{10}$</td>
<td>2.1</td>
<td>2.1</td>
</tr>
<tr>
<td>$d_{78}$</td>
<td>1.4</td>
<td>2.0</td>
</tr>
<tr>
<td>$d_{23}$</td>
<td>1.4</td>
<td>1.7</td>
</tr>
<tr>
<td>$d_{64}$</td>
<td>1.1</td>
<td>1.9</td>
</tr>
</tbody>
</table>

**SA** | **RA**
### No-Random-Accesses Algorithm (NRA)

- **Sequential accesses (SA) only**
- **Worst-case memory consumption** $O(m \times n + k)$

**No-Random-Accesses Algorithm (NRA):**

1. Scan index lists (e.g., round-robin)
2. Consider $d = cdid(i)$ in posting list for $t_i$
3. $high(i) = cscore(i)$
4. $eval(d) = eval(d) \cup \{i\}$  // where have we seen $d$?

- $worst(d) = aggr\{\text{score}(t_j, d) | j \in eval(d) \}$
- $best(d) = aggr\{worst(d), aggr\{high(j) | j \notin eval(d) \}\}$

**Top-1:***

<table>
<thead>
<tr>
<th>worst</th>
<th>best</th>
</tr>
</thead>
<tbody>
<tr>
<td>$d_{10}$ : 2.1</td>
<td>2.1</td>
</tr>
<tr>
<td>$d_{78}$ : 1.4</td>
<td>2.0</td>
</tr>
<tr>
<td>$d_{23}$ : 1.4</td>
<td>1.7</td>
</tr>
<tr>
<td>$d_{64}$ : 1.1</td>
<td>1.9</td>
</tr>
</tbody>
</table>

**ua** = 2.0
Combined Algorithm (CA)

• **Balanced SA/RA Scheduling:**
  
  • define **cost ratio** \( r = \frac{C_{SA}}{C_{RA}} \) (e.g., based on statistics for execution environment, typical values \( C_{RA}/C_{SA} \sim 100 - 10,000 \) for hard disks)
  
  • run **NRA** (using SA only) but perform one RA every \( r \) rounds (i.e., \( m \times r \) SAs) to look up the unknown scores of the **best candidate** that is not in the current top-\( k \)

• **Cost competitiveness** w.r.t. “optimal schedule” (scan until \( \text{aggr}\{ \text{high}(i) \} \leq \text{min}\{ \text{best}(d) \mid d \in \text{final top-}k \} \)), then perform RAs for all \( d' \) with \( \text{best}(d') > \text{min}_k \): \( 4 \times m + k \)
TA / NRA / CA Instance Optimality

• **Definition**: For class of algorithms $A$ and class of datasets $D$, algorithm $A \in A$ is **instance optimal** over $A$ and $D$ if

\[ \forall A' \in A \ \forall D \in D : \ cost(A, D) \leq c \cdot cost(A', D) + c' \]

(i.e., $cost(A, D) \in \mathcal{O}(cost(A', D))$)

• **TA is instance optimal** over all top-$k$ algorithms based on random and sequential accesses to $m$ lists (no “wild guesses”)

• **NRA is instance optimal** over all top-$k$ algorithms based on only sequential accesses

• **CA is instance optimal** over all top-$k$ algorithms based on random and sequential accesses and given cost ratio $C_{RA}/C_{SA}$

• **Full details**: [Fagin et al. ’03]

\[ A_0 \leq A \leq D_0 \leq D \]
Implementation Issues for Threshold Algorithms

- Limitation of asymptotic complexity
  - \( m \) (# lists), \( n \) (# documents), \( k \) (# results) are important parameters
- Priority queues
  - straightforward use of heap (even Fibonacci) has high overhead
  - better: periodic rebuilding of queue with partial sort \( O(n \log k) \)
- Memory management
  - peak memory usage as important for performance as scan depth
  - aim for early candidate pruning even if scan depth stays the same
7. Query Processing with Importance Scores

- Focus on score combining **textual relevance** (\(rel\)) (e.g., TF*IDF) and **global importance** (\(imp\)) (e.g., PageRank)

\[
score(q, d) = imp(d) + rel(q, d)
\]

with **normalization** \(imp(d) \leq a\) and \(rel(q, d) \leq b\) and \(a + b \leq 1\)

- Keep posting lists in descending order of **global importance**

  \[
  high(i) = imp(cdid(i)) + b \\
  high = \max\{ high(i) \mid i = 1 \ldots |q| \} + b \\
  \]

  Stop scanning \(i\)-th posting list when \(high(i) < min_k\) (i.e., minimal score in top-\(k\))

  Terminate when \(high < min_k\)

  effective when combined score is dominated by \(imp(d)\)

- **First-\(k\)' heuristic**: Scan all posting lists until \(k' \geq k\) documents have been seen in all lists, so that their combined score is known

- **Full details**: [Long and Suel '03]
8. Query Processing with Champion Lists

**Idea:** In addition to full posting lists $L_i$ sorted by $imp(d)$, keep short "champion lists" sorted (aka. "fancy lists") $F_i$ that contain docs $d$ with the highest values of $score(t_i, d)$ and sort these lists by $imp(d)$

**Champions First-$k'$ heuristic:**

Compute total score for all docs in $\bigcap F_i (i = 1 \ldots |q|)$ and keep top-$k$ results

$cand = \bigcup F_i - \bigcap F_i$

**for each** $d \in cand$ **do**

compute partial score of $d$

scan full posting lists $L_i (i = 1 \ldots |q|)$

**if** $cdid(i) \in cand$ **then**

add $score(t_i, cdid(i))$ to partial score of $cdid(i)$

**else**

add $cdid(i)$ to $cand$ and set its partial score to $score(t_i, cdid(i))$

terminate the scan when we have $k'$ documents with complete scores

**Full details:** [Brin and Page ’98]
Summary of V.3

• **Query Type**
  determines usefulness of optimizations (e.g., skip pointers)

• **Term-at-a-Time** and **Document-at-a-Time**
  for holistic query processing

• **WAND**
  for top-$k$ query processing on document-ordered posting lists

• **Buckley’s Algorithm**
  for top-$k$ query processing on scored-ordered posting lists

• **Fagin’s Threshold Algorithms**
  top-$k$ query processing with, without, or with some RAs
Additional Literature for V.3


- **A. Broder, D. Carmel, M. Herscovici, A. Soffer, J. Zien**: *Efficient query evaluation using a two-level retrieval process*, CIKM 2003

- **C. Buckley and A. Lewit**: *Optimization of Inverted Vector Searches*, SIGIR 1985


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