Skewed Partial Bitvectors for List Intersection

Andrew Kane and Frank Wm. Tompa
University of Waterloo

arkane@cs.uwaterloo.ca

SIGIR - July 8th 2014
Outline

• Introduction to search engines.
• List Intersection - compression, skips, bitvectors.
• Document reordering - URL vs. terms-in-document.
• Partial bitvectors - semi-bitvectors with skewed groups.
• Conclusions.
Search Engine Query Processing

Index

Query | Lookup | Intersect | Rank top-k | Expand | Result
     | terms  | encodings | list      | short list | metadata

AND
OR
Weak-AND
Phrase
Proximity

Early Termination
Pruning
Search Engine Query Processing

Index

Query | Lookup | Intersect | Rank top-k | Expand | Result

terms | encodings | list | short list | metadata

space
time

AND
OR
Weak-AND
Phrase
Proximity

Early Termination
Pruning
Search Engine Query Processing

Index

<table>
<thead>
<tr>
<th>Query</th>
<th>Lookup</th>
<th>Intersect</th>
<th>Rank top-k</th>
<th>Expand</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>terms</td>
<td>encodings</td>
<td>list</td>
<td>short list</td>
<td>metadata</td>
<td></td>
</tr>
</tbody>
</table>

AND
OR
Weak-AND
Phrase
Proximity

Early Termination
Pruning
Data for List Intersection

<table>
<thead>
<tr>
<th>Terms</th>
<th>Documents</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

- Inverted index using document level postings lists.
List Intersection

- To intersect, find the query term lists.
execute list intersection (small to large).

we run pairwise (term-at-a-time) processing of conjunctive-AND with lists ordered by document ID (allows merge).
List Encoding

Bitvector: X X X X X X X X X

Uncompressed: $p_0, p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8, p_9, \ldots$

Compressed: $\Delta_0, \Delta_1, \Delta_2, \Delta_3, \Delta_4, \Delta_5, \Delta_6, \Delta_7, \Delta_8, \Delta_9, \ldots$

Compressed + skips: $s_0, s_1, \ldots, \Delta_0, \Delta_1, \Delta_2, \Delta_3, \Delta_4, \Delta_5, \Delta_6, \Delta_7, \Delta_8, \Delta_9, \ldots$

- Bitvectors are fast.
- Uncompressed lists are fast, but waste probes.
- Compressed lists have no random access.
List Encoding

- Very Large
  - Bitvector: X X X X X X X X

- Large
  - Uncompressed: p₀, p₁, p₂, p₃, p₄, p₅, p₆, p₇, p₈, p₉, ...

- Slow
  - Compressed: Δ₀, Δ₁, Δ₂, Δ₃, Δ₄, Δ₅, Δ₆, Δ₇, Δ₈, Δ₉, ...

- Compressed+skips: S₀, S₁, Δ₀, Δ₁, Δ₂, Δ₃, Δ₄, Δ₅, Δ₆, Δ₇, Δ₈, Δ₉, ...

- Can also use a hybrid bitvector approach:
  - Use bitvectors if freq. > F and compressed lists for others [Culpepper and Moffat 2010].
List Encoding

- **Add (large overlaid) skips to compressed lists** [new]
• Renumber the documents (columns) for improvements.
Document Ordering (url)

- URL ordering ($\approx$ similarity) gives **tight clustering**.
• Terms-in-document (td) ordering gives skewed clustering.
Combine Using Groups [new]

- Group documents by terms-in-document to get skewed clustering.
- Order within each group by URL to get tight clustering.
- Call this td-g#-url ordering.
Skew still remains in td-g3-url ordering.
Partial Bitvectors [new]

- Semi-bitvector data structure [new]:
  - Encode the front of a list as a bitvector.
  - Encode the remainder using skips and delta compression.
- Stores more postings in bitvectors (faster) for given space (so more efficient).
• Semi-bitvectors with grouping are fast and compact.
Experiments

- Conjunctive-AND list intersection in memory.
- Space = encoded size of lists (no dictionary).
- Time = list intersection (no lookup in dictionary).
- Using GOV2 dataset (426GB) and 5000 corpus queries (4.1 terms per query).
  - Original order ≈ random order.
Experimental Results (orig)

- Bitvectors are very fast and skips (X=256) help other lists.
Experimental Results (url)

- Bitvectors+skips still faster after reordering.
Experimental Results (td-g8-url)

- Improvements using semi-bitvectors with 8 groups.
Conclusions

- **Skips** can improve hybrid bitvector approaches.
- **Grouping** can combine skewed and tight clustering.
- **Semi-bitvectors with grouping** improves space and time:
  - More postings in bitvectors for given space.
  - Smaller deltas for compression.
  - Better locality of access and shared decoding.

- Future work:
  - Combine with ranking based systems.
Thank you.

Questions?

/* Comments */

Andrew Kane: arkane@cs.uwaterloo.ca

(Student travel grant generously provided by SIGIR)
Existing Ordering Approaches

- Rank based – PageRank
- Matrix manipulation – block diagonal
- Document size
- Content similarity – similarity clustering, TSP
- Metadata – URL
- Hybrid – url.server.suffix-td-url
• Terms-in-document count for GOV2 dataset, split by number of postings into three groups, produces skew.
  • Area under the curve is equal for each partition.
Entropy for Grouping

- Grouping can make the data more compressible.
- Degradation is slow (entropy of $td = 5.07$)
Postings in Bitvectors

- Grouping and semi-bitvectors allows more postings to be stored in bitvectors
Semi-bitvectors with conjunctive-AND are faster than these ranking based systems (from published results with similar hardware) on GOV2.
Semi-bitvectors in Ranking

- Pre-filter – run AND first then ranking.
- Sub-document pre-filter – AND over windows.
- High density filters – store dense regions as bitvectors, since low rank information.
- Query specific filter – dynamically pick terms to use semi-bitvectors.
- Guided processing – execute on subset of documents using AND to decide how to process query.