Outline

• Introduction to search engines.
• Distribution by document size.
• Experiments:
  • Space improvement.
  • Runtime improvements.
• Applications in practice.
## Search Engine Query Processing

<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
Search Engine Query Processing

<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
List Intersection

- Pairwise list intersection
  - Here we use conjunctive-AND with lists ordered by document ID.
Document Ordering

- Renumbering the documents affects space-time efficiency.
  - **Best is URL ordering** (similar to clustering).
  - Document size ordering (terms-in-document or td) is worse than URL ordering.
    - So, people typically ignore td ordering.
  - Random ordering is used as a base of comparison.
Early Termination

• When list intersection will produce lots of results:
  • Store each list in impact order (usually frequency), rather than by document ID.
  • Process only fronts of lists (early termination).
  • Use accumulators to combine lists.

• Impact ordering can outperform URL ordering.
Search Engine Query Processing

Scale by Distribution

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

AND
OR
Weak-AND
Phrase
Proximity

Early Termination
Pruning
Document Distribution

- How do you distribute documents to partitions?

Figure from Baeza-Yates et al. ICDE 2007
Random distribution is normally used:
- Balanced distribution of query work and index size.
- We refer to this as rand-p.

Claim:
Document size distribution improves performance:
- Benefits to index size and query resource usage.
- Balancing requires tuning of the partition cutoff points.
- We measure size by # terms in document.
- We refer to this as td-p.
Within Partitions

- Can use any ordering within the partitions.
  - We use random ordering for our tests to avoid bias, so we compare rand-p-rand vs. td-p-rand.
  - Using URL ordering produces similar types of improvement (i.e., td-p-url is better than url-p-url).
  - Future work: compare td-p-impact and rand-p-impact.
Experiments

• Conjunctive-AND list intersection in memory.
• Three partitions with equal number of postings.
• Sum index space and query runtime over partitions.
• Setup:
  • Using GOV2 dataset (426GB) and 5000 corpus queries (4.1 terms per query).
Document Size

- Terms-in-document count for GOV2 dataset, split by number of postings into three partitions, produces skew.
- Area under the curve is equal for each partition.
Density in Partitions

• Skew in list density (for queries):
  \[ D_{\text{Large}} \quad D_{\text{medium}} \quad D_{\text{Small}} \]
  Smallest list density 2.39% 0.98% 0.23%

• Skew in result density is even larger:
  \[ D_{\text{Large}} \quad D_{\text{medium}} \quad D_{\text{small}} \]
  Result list density 0.50% 0.11% 0.02%

• So, exploit this skewed density.
Compressed List Encoding

- We use simple16 compression: header+data = 4+28 bits
Results for Compressed Lists

- Encoding as compressed lists of deltas (simple16):
  - rand-p-rand: 7.54 bits/posting.
  - td-p-rand: 6.70 bits/posting.

- Space improvement of 11.1%.

- Runtime essentially the same.
Adding Skips to Encoding

Skips: 20 37 ...

Compressed: X 5 5 2 8 Y 1 7 3 6 ...

• Tuning: number of postings skipped.
Benefits with Skips

- Skew from terms-in-document distribution:
  - In small-document partitions:
    - Reduces density of intermediate results.
    - Therefore, skips more effective.
  - In large-document partitions:
    - Increases density of postings.
    - Therefore, cache line clustering (locality of access).
    - Amortized costs to decode a block.
Results for Skips

![Graph showing performance results for Skips with different space and time metrics. The graph compares S16+skips(rand−p−rand) and S16+skips(td−p−rand).](image)
Bitvector Encoding

- Tuning: use bitvectors if freq. > F and compressed lists for others [Culpepper and Moffat 2010].
Benefits with Bitvectors

- Bitvectors used more effectively:
  - Density threshold $F$ applied to each term independently in each partition.
  - Therefore, more bitvectors (green) in large-document partitions.
Results for Bitvectors

![Graph showing time (ms/query) vs. space (bits/posting) for different space utilization rates. The graph compares two methods: S16+bitvectors(rand−p−rand) and S16+bitvectors(td−p−rand). The data points are at space utilization rates of 1/4, 1/8, 1/16, 1/24, 1/32, 1/48, and 1/64.](image)
Distribution in Practice

- Use a hierarchy of distribution/ordering mechanisms in practice, for example:
  - Tier documents by global relevance (e.g., PageRank).
  - URL domain distribution (e.g., .gov) within a tier.
  - Document Size Distribution within a domain.
  - Order by URL or impact within partition.
Conclusions

• We have shown that document size distribution improves space and time:
  • Compression of postings lists.
  • Locality of access inside structures.
  • Performance of skips and bitvectors.

• Document size distribution is broadly applicable.

• Future work:
  • Compare td-p-impact and rand-p-impact.
Thank you.

Questions?

/* Comments */

Andrew Kane: arkane@cs.uwaterloo.ca
Ranking

- **Direct improvements:**
  - Delta compression and skips are often used in ranking systems.

- **Expected improvements:**
  - Locality of access from increased density of lists.
  - Sparse intermediate results.
  - Structures/processing that adapts to each partition.
Potential Improvements

• Within a partition:
  • Tune algorithms in each partition to fit the data in partition.

• Across partitions:
  • Run on subset of partitions to decide on subsequent processing. For example, decide on AND vs. Weak-AND processing for other partitions.