In-Memory Text Search Engines

Peter Sanders,
Frederik Transier
Demands and Goals of Text Search Engines

- Locating words in a huge number of documents.

- Important Operations: AND queries, phrase queries, document reporting.

- Using main-memory for maximum query performance.
Inverted Index

- Unique IDs for:
  - documents
  - terms

- A document becomes a list of term IDs.

- Inverted Index:
  For each term we store a list of document IDs.

- Positional Inverted Index:
  Positions list for each term / document pair.
Intersectio*n Algorithms: Randomized Inverted Indices

- Randomized Inverted Index: document IDs are assigned randomly. (e.g. using pseudo-random permutation)

- Two-level data structure:
  - Split the range of document IDs into buckets based on their most significant bits.
  - **Lookup-table**: direct access to the first value of a bucket

Inverted List with lookup table

(lookup table size: $\lceil \log_2(n/B) \rceil$)
Intersection Algorithms: Randomized Inverted Indices

`Lookup` is an intersection algorithm that runs on this data structure:

Function `lookup(M,N)`

```
O := {} // output
i := -1 // current bucket key (now a dummy)

foreach d ∈ M do // unpack M
    h := d >> k_N // bucket key
    l := d & (2^{k_N}-1) // least significant bits
    if h > i then // a new bucket
        i := h // set current bucket
        j := t[i] // get start position
        e := t[i+1] // get end position
        while j < e do // bucket not exhausted
            l' := N[j] // unpack if necessary
            if l ≤ l' then
                if l = l' then O := O ∪ {d}
                break while
            j++
    return O
```

Lookup runs in expected time \(O(m+\min\{n,Bm\})\)
Experiments: Space Consumption (WT2g)

No significant differences between $\Delta$-bitc. and $\Delta$-escaped for the rand. representation.

$\Delta$-bitc. does not work well for the det. representation.

Escaping can exploit nonuniformity of input.
Experiments: Performance of *Lookup* (WT2g)

- Large buckets are bad for small ratios...
- ...but good for nearly equal lengths.
- Bucket size 8 seems to be a good compromise.
- bit-compressed Δ-encoding
Experiments: Performance Comparison (WT2g)

Zipper, skipper, and lookup are all very good for lists of similar lengths.

Lookup is best up to ratio close to one.
Experiments: Space-time Tradeoff of Encodings (WT2g.s)

- Performance loss of $\Delta$-encoding is negligible low for rand. representation.
- Escaping requires perceivable more time.
- Clearly different running times for det. representation.
Experiments: Impact of Randomization on \textit{Lookup}

Randomization gives theoretical performance guarantees, but in practice deterministic data often outperforms randomization.

\textit{Lookup} is also a good heuristics for non-randomized data.
## Suffix Arrays

- Suffix Arrays as full-text index: concatenate all text documents.
- Phrase search: A home match for SAs.
- AND searches:
  - search for terms
  - Intersect the occurrence lists.
- Document reporting
- Compressed or distinct SAs are available from the Pizza&Chili website:
  
  http://pizzachili.di.unipi.it

Suffix array of „abracadabra“
**Performance of different SAs on WT2g 1-50000**

<table>
<thead>
<tr>
<th></th>
<th>CSA</th>
<th>CCSA</th>
<th>SSA2</th>
<th>AF-index</th>
</tr>
</thead>
<tbody>
<tr>
<td>size [MB]</td>
<td>230.9</td>
<td>500.9</td>
<td>302.6</td>
<td>279.3</td>
</tr>
<tr>
<td>compression</td>
<td>0.64</td>
<td>1.39</td>
<td>0.84</td>
<td>0.77</td>
</tr>
<tr>
<td>indexing time [min]</td>
<td>9.3</td>
<td>11.5</td>
<td>8.7</td>
<td>23.0</td>
</tr>
<tr>
<td>peak mem usage [GB]</td>
<td>3.2</td>
<td>3.1</td>
<td>2.1</td>
<td>3.1</td>
</tr>
</tbody>
</table>

**Graphs:**
- **AND** query performance
- **Phrase** query performance
Modular Design of the Compressed Inverted Index (CII)

Documents → Preprocessor → Dictionary → Text delta

- Preprocessor
  - Normalized documents (term ID/doc ID/pos)
  - Terms

- Dictionary
  - Differences between documents and their normalized versions

- Text delta
  - Bags of words
  - Positional index
  - Document-grained inverted index
CII: Document-grained Inverted Index

Direct stored doc IDs

Golomb coded list of doc ID deltas

Two-Level Lookup lists
Two-level data structure, refinement of [Sanders and Transier, 2007]:

- Split the range of document IDs into buckets based on their most significant bits.
- **Lookup-table**: direct access to the first value of a bucket
- **Rank information**: number of values smaller than the contents of a bucket
- **Variable Golomb coding**: estimating the average of each bucket.
CII: Positional Index

- Direct stored positions
- Golomb coded list of doc ID deltas
  - Single doc
  - Multi doc / single pos
  - Multi doc / multi pos
- Bitcompressed lists (indexed by ranks)
- Two-Level list (top-level indexed by ranks)
Querying the Compressed Inverted Index

- **AND Query:** Intersection of inverted lists in increasing order of list lengths.

```
(((∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∩∅
CII: Document Reporting with Text Delta

- **Document 5**: Bag of words → Document index → Positional index
  - List of normalized terms
  - List of term escapes
  - Escape dictionary
  - List of original terms
  - Document index:
    - List of term separators
    - Next 8 terms are separated by spaces.
    - Next item a term

- **Document Index**:
  - 5
  - 7
  - 0
  - 8
  - 9
  - 4
  - 1
  - 3
  - 9
  - 6
  - 5
We have implemented all algorithms using C++.

One core of an Intel Core 2 Duo E6600, clocked at 2.4 GHz with 2 x 2MB L2 cache and 4 GB main memory.

openSuSE 10.2 (kernel 2.6.18), gcc 4.1.2 (-O3)

Timing with PAPI 3.5.0
Experiments: Test Data

- Real world instance: first 50,000 docs of WT2g.
- Pseudo real-world queries:
  - AND and phrase queries.
  - Selecting random hits.
  - Query lengths: 1-10 terms.
## Indexing: Space requirements

<table>
<thead>
<tr>
<th></th>
<th>CII</th>
<th>CSA</th>
</tr>
</thead>
<tbody>
<tr>
<td>dictionary</td>
<td>23.9</td>
<td></td>
</tr>
<tr>
<td>document Index</td>
<td>32.4</td>
<td></td>
</tr>
<tr>
<td>positional Index</td>
<td>126.3</td>
<td></td>
</tr>
<tr>
<td>bag of words</td>
<td>25.1</td>
<td></td>
</tr>
<tr>
<td>suffix array</td>
<td></td>
<td>230.8</td>
</tr>
<tr>
<td>doc bounds</td>
<td></td>
<td>0.1</td>
</tr>
<tr>
<td>sum [MB]</td>
<td>206.1</td>
<td>230.9</td>
</tr>
<tr>
<td>text delta [MB]</td>
<td>108.7</td>
<td></td>
</tr>
<tr>
<td>sum + text delta [MB]</td>
<td>314.8</td>
<td></td>
</tr>
<tr>
<td>input size (norm.)</td>
<td>412.8</td>
<td>(360.5)</td>
</tr>
<tr>
<td>compression</td>
<td>0.76</td>
<td>(0.57)</td>
</tr>
<tr>
<td>indexing time [min]</td>
<td>5.6</td>
<td>(5.1)</td>
</tr>
<tr>
<td>peak mem usage [GB]</td>
<td>0.7</td>
<td>3.2</td>
</tr>
</tbody>
</table>
Experiments: Average AND Query Time

- Average \(<1\) ms for all lengths.
- 3-4 orders of magnitude slower than CII.
Experiments: How many % AND queries take longer than $t$?

- All queries are done in less than 3.6 ms.
- 35 s in worst case.
- Query length of 2 terms.

All queries are done in less than 3.6 ms.

35 s in worst case

Query length of 2 terms.
Experiments: Average Phrase Query Time

CSA is marginal faster.
(factor 0.94 – 0.62, < 1ms absolute)

CII is more than 20 times faster than CSA.
Experiments: How many % phrase queries take longer than t?

Query length of 2 terms.

$t < 190$ ms for all queries.

Factor 24 apart.
Experiments: Document Reporting

Assuming a disk access latency of 5ms, load from disk would be faster for documents > 32 KB.

Bandwidth of CSA is about 5 times smaller

Bandwith of CSA is about 5 times smaller

6-8 MB/s

data rate [MB/s] vs. text size [KB]
Outlook & Future Work

- Bag of words: adaptive coding scheme?
- Compression of the dictionary.
- Speeding up the most expensive phrase queries.
- Construction time?
- Fast updates?
Thank You for Your attention!
CII on the complete WT2g / WT2g.s

- Short queries are slower on WT2g.s, as there are more results.
  - Large queries are faster, as they benefit from more lookup lists.

Phrase queries are faster on WT2g.s, because the position lists are shorter.