How we Made it to 1st Place on the Leaderboard

Henrik Mühe
henrik.muehe@in.tum.de

Florian Funke
florian.funke@in.tum.de

The Final Leaderboard

<table>
<thead>
<tr>
<th>Team</th>
<th>Small (sec)</th>
<th>Big (sec)</th>
<th>New (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Campers (TUM)</td>
<td>0.081</td>
<td>1.938</td>
<td>7.515</td>
</tr>
<tr>
<td>Rotafortune (St. Petersburg University)</td>
<td>0.158</td>
<td>1.969</td>
<td>9.394</td>
</tr>
<tr>
<td>matematik (Bielefeld University)</td>
<td>0.065</td>
<td>1.507</td>
<td>10.343</td>
</tr>
<tr>
<td>gfr</td>
<td>0.137</td>
<td>2.100</td>
<td>11.795</td>
</tr>
<tr>
<td>pharos (Helsinki University)</td>
<td>0.585</td>
<td>2.320</td>
<td>12.794</td>
</tr>
<tr>
<td>StrongAccept (Tsinghua University)</td>
<td>0.396</td>
<td>3.019</td>
<td>12.848</td>
</tr>
<tr>
<td>null</td>
<td>22.483</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>ePetra</td>
<td>30.927</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>null</td>
<td>43.174</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>null</td>
<td>43.786</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Inherent Optimizations

1. Document deduplication, trivial
2. Query word clustering, NOT trivial

Example

Q1: justin bieber
Q2: justin timberlake
Q3: justin time

Algorithm

- For each query word, determine "skip words"
- Incrementally remove skip words, periodically recompute
- Skip vector for all deactivate words

Low-Overhead Filtering

Use filters to determine if two words can be within Edit/Hamming distance $d$.

Challenge: Filter must be substantially faster than invoking the metric itself

Our filters:

1. Length: $|\text{word}_a| - |\text{word}_b| \leq d \implies$ possible match
2. Frequency Histogram (full size and folded):

   "abb" Delta $\Delta$ "abbab"

   \[
   \begin{align*}
   b=2 & : 1 \\
   a=1 & : 2 \\
   z=0 & : 2 \\
   \sum & = 2
   \end{align*}
   \]

   $\Delta \leq 2d - |\text{word}_a| - |\text{word}_b|$

Haar Wavelet Index

Query Words

- wavelets ordered by coefficients
- offsets points to first string with this wavelet
- query words ordered by wavelets

Doc Words

- doc words ordered by wavelets
- offsets points to first string with this wavelet
- wavelets ordered by coefficients

Match Caching

Observation

People make the same types again and again, e.g. "calendar" vs "calender".

Idea

- Cache misspelled (document) words with matching query words (for each metric and each distance).
- Probe document's hash table with misspelled word.
- Often saves the iteration through entire document.

The Metrics

- Exact Matching: Test if two words are equal
- Hamming Distance: Number of positions that differ between two words of the same size
- Levenshtein Edit Distance: Minimum number of insert/delete/substitute operations between two words

The API

- startQuery(qId, query, type, distance)
- searchQuery(qId)
- matchDocument(docId, document)
- setNextAvailableResult(Result)

Parallelism & Concurrency

matchDocument(char* doc) -> strcpy() -> spmm() -> return

Task: MatchDocument
- Tokenization and Preparation
- spmm()
- Exact Matching
- join()
- Result compilation
- enqueue(Result)

Task: Edit Distance
- Hierarchically expose parallelism
- No unnecessary synchronization

Blazing-Fast Metric Computation

Hamming Distance
- SIMD instructions
- Improved version over streaming string SSE4

Hamming Distance:

Minimum number of differences between two words of the same size

Exact Matching:

Number of positions that differ between two words of the same size

Levenshtein Edit Distance:

Minimum number of insert/delete/substitute operations between two words

Matching queries

Search for two words

Match Caching

Exact Matching: Test if two words are equal