Efficient Exact Similarity Join Algorithms

Wei Wang
University of New South Wales
Australia
http://www.cse.unsw.edu.au/~weiw

Roadmap
- Motivation
- Problem Definition and Scope
- Approximate Similarity Join Algorithms
- Conclusions

Deduplication

On one end, a winded Pete Sampras tried to summon enough energy to give the New York fans another memorable win to talk about on the subway ride home. On the other side, Roger Federer wore a sly grin like he knew age was about to catch up to the former world No. 1: the man who owns the record of 14 Grand Slams.

Roadmap
- Motivation
- Problem Definition and Scope
- Approximate Similarity Join Algorithms
- Conclusions

More Applications
- For Web search engines:
  - Perform focused crawling
  - Increase the quality and diversity of query results
  - Identify spams.
- For Web mining:
  - Perform document clustering
  - Find replicate Web collections
  - Detect plagiarism

Data Mining Applications
- UNIPROT identified as 2 clusters in UNIREF-90
- DBScan
- Collaborative filtering

FACT: MEDLINE DB has ~11M records

Plagiarism in MEDLINE databases

Acta Astronaut (Mar-Apr-1975)
Tuberculosis (Edinb); (2001)

Acta Astronaut (Mar-Apr-1975)
Roadmap

- Motivation
- Problem Definition and Scope
- Approximate Similarity Join Algorithms
- Epilogue

7/10/2008

Similarity Join

- Input
  - two sets of records: R and S
  - a similarity function: \( \text{sim}(r, s) \)
  - a threshold: \( t \)
- Output
  - all pairs of object \( r \in R, s \in S \), such that \( \text{sim}(r, s) \geq t \)
- Variations
  - \( d(r, s) \leq t \) to match customers' names.

7/10/2008

Focus on Self Join

7/10/2008

Similarity/Distance Functions

- We mainly consider objects as texts / (multi-)sets
- We first focus on the overlap similarity function
  - Overlap( set1, set2 ) = set1 \( \cap \) set2
- Strings to sets
  - Tokenization + Recurring token as a new token
  - Overlap( str1, str2 ) = \# of common tokens in both strings
- Consider others (Jaccard, cosine, edit-distance) later

7/10/2008

Example

<table>
<thead>
<tr>
<th>RID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Database System Concepts</td>
</tr>
<tr>
<td>2</td>
<td>Database Concepts Techniques</td>
</tr>
<tr>
<td>3</td>
<td>Database System Programming Concepts Oracle Linux</td>
</tr>
<tr>
<td>4</td>
<td>Database Programming Concepts Illustrated</td>
</tr>
<tr>
<td>5</td>
<td>System Programming Concepts Oracle Linux</td>
</tr>
</tbody>
</table>

\( r = 3 \)

\( \{1, 3, 2, 2, 2\} \)

7/10/2008

Naïve Algorithm

<table>
<thead>
<tr>
<th>RID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Database System Concepts</td>
</tr>
<tr>
<td>2</td>
<td>Database Concepts Techniques</td>
</tr>
<tr>
<td>3</td>
<td>Database System Programming Concepts Oracle Linux</td>
</tr>
<tr>
<td>4</td>
<td>Database Programming Concepts Illustrated</td>
</tr>
<tr>
<td>5</td>
<td>System Programming Concepts Oracle Linux</td>
</tr>
</tbody>
</table>

7/10/2008
**Not-so-naïve Algorithm**

- Observation: (?; 6) should not have been compared
  - If x and y have no common token, they won’t be in the result
- Idea: Use inverted index to consider promising candidate pairs only

<table>
<thead>
<tr>
<th>RID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Database System Concepts</td>
</tr>
<tr>
<td>2</td>
<td>Database Concepts Techniques</td>
</tr>
<tr>
<td>3</td>
<td>Database System Programming Concepts Oracle Linux</td>
</tr>
<tr>
<td>4</td>
<td>Database Programming Concepts Illustrated</td>
</tr>
<tr>
<td>5</td>
<td>System Programming Concepts Techniques Oracle Linux</td>
</tr>
<tr>
<td>6</td>
<td>Harry Potter and the Sorcerer’s Stone</td>
</tr>
</tbody>
</table>

**Inverted Index**

- Conceptually, an inverted index has an inverted list for each token to be indexed from the document collection
  - An inverted list is just an sorted array of document identifiers (in our case, RIDs) such that the token appears in the corresponding document

<table>
<thead>
<tr>
<th>RID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Database System Concepts</td>
</tr>
<tr>
<td>2</td>
<td>Database Concepts Techniques</td>
</tr>
<tr>
<td>3</td>
<td>Database System Programming Concepts Oracle Linux</td>
</tr>
<tr>
<td>4</td>
<td>Database Programming Concepts Illustrated</td>
</tr>
<tr>
<td>5</td>
<td>System Programming Concepts Techniques Oracle Linux</td>
</tr>
</tbody>
</table>

**Probe Count**

<table>
<thead>
<tr>
<th>RID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Database System Concepts</td>
</tr>
<tr>
<td>2</td>
<td>Database Concepts Techniques</td>
</tr>
<tr>
<td>3</td>
<td>Database System Programming Concepts Oracle Linux</td>
</tr>
<tr>
<td>4</td>
<td>Database Programming Concepts Illustrated</td>
</tr>
<tr>
<td>5</td>
<td>System Programming Concepts Techniques Oracle Linux</td>
</tr>
</tbody>
</table>

1. Consider each term, token, in the current record and retrieve their inverted lists through the index
2. Merge these RID-lists
3. Repeat step 1 & 2 for every record

**Problems**

- Still too many comparisons
  - tokens that appears in many documents not only results in large inverted lists, but also slow down the computation

<table>
<thead>
<tr>
<th>RID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Database System Concepts</td>
</tr>
<tr>
<td>2</td>
<td>Database Concepts Techniques</td>
</tr>
<tr>
<td>3</td>
<td>Database System Programming Concepts Oracle Linux</td>
</tr>
<tr>
<td>4</td>
<td>Database Programming Concepts Illustrated</td>
</tr>
<tr>
<td>5</td>
<td>System Programming Concepts Techniques Oracle Linux</td>
</tr>
<tr>
<td>6</td>
<td>Database of Respiratory Diseases</td>
</tr>
</tbody>
</table>

**Additional Notes**

- Several other optimizations exist (inverted techniques, boosting, etc.)
Prefix Filtering

- Establish an upper bound of the overlap between two sets based on part of them

\[ \text{if } \text{t=4, overlap(player1, player2) < t} \]
\[ \text{or upperbound(overlap(player1, player2)) = t-1} \]

What's the maximum possible number of cards held by both players (denomination not considered)?

Prefix Filtering for Absolute Overlap Constraint

- Constraint:
  - \( \text{overlap}(S_1, S_2) \geq t \)

- Pre-requisite:
  - All sets sorted in a global order
  - Usually the decreasing idf order

- Preprocessing:
  - Set \( S \text{ prefix(S)} \), s.t., \( |S| - (t - 1) \)

- Test:
  - If \( \text{Prefix}(S_1) \cap \text{Prefix}(S_2) = \emptyset \), then \( \text{overlap}(S_1, S_2) < t \)
  - i.e., \( (S_1, S_2) \) can be filtered out

Overlap Self Similarity Join Based On Prefix Filtering

- for each \( S_j \in S \) // nested loop
  - Candidates = \( \emptyset \)
  - prefix-len = \( |S_j| - (t - 1) \)
  - for \( i=1 \) to prefix-len // go thru prefix
    - \( w = S_j[i] \)
    - for each \( S_k \in \text{Inverted-list}(w) \)
      - Candidates = Candidates \( \cup \) \( S_k \)
    - \( \text{Verify}(S_j, \text{Candidates}) \)

All-Pairs algorithm: \( \text{Prefix-Sim}(x, y_i) \)
Motivation for ppjoin [Chuan et al, WWW08]

- All-Pairs is still not fast enough
  - It generates many candidates
  - It needs to perform many verifications
- Can be made even more efficient by further exploiting the global ordering
  - Record the position of the tokens in the prefix \( \rightarrow \) ppjoin
  - Probe the tokens in the suffixes \( \rightarrow \) ppjoin+
  - Effective for Jaccard/cosine similarity constraints

Consider Jaccard Similarity

- Jaccard similarity
  \[ J(x, y) \geq t \iff \Omega(x, y) \geq \alpha - \frac{t}{1 + t} (|x| + |y|) \]
  \[ \text{prefix-len}(x) = |x| - |t| \cdot |x| + 1 \]

How Positional Information Helps/1

- Derive an upper bound of the overlap based on position information in the prefixes
  \[ \text{overlap}(x, y) \leq \min\{|x|, |y|\} - \text{prefix-len}(x) \]

How Positional Information Helps/2

- Also useful in verification

ppjoin+ /1

- Can position information be used to the suffixes?
  \[ \text{overlap}(x, y) \leq 15 \]
  - \( <x, y> \) is not a candidate pair for \( t \leq 0.8 \)
  - Overlap \( \geq 0.8 \) must be \( \geq 16 \)

Relationships among Similarity/Distance Functions

- Jaccard similarity
  \[ J(x, y) - |x \cap y| / |x \cup y| \]
  \[ J(x, y) \geq t \quad \iff \quad \Omega(x, y) \geq \alpha - \frac{t}{1 + t} (|x| + |y|) \]
  \[ J(x, y) \geq t \quad \Rightarrow \quad |x \cap y| \leq \frac{|x| \cdot |y|}{t} \quad (\text{wlog. if } |y| \leq |x|) \]

- Cosine similarity
  \[ \text{sims}(x, y) \geq t \quad \iff \quad \gamma(x, y) \geq \sqrt{\text{sims}(x, y)} \]
  \[ \cos(x, y) \geq t \quad \Rightarrow \quad r^2 \cdot |x| < |y| \quad (\text{wlog. if } |y| \leq |x|) \]
ppjoin+ /2
- Apply multiple probes in a divide-and-conquer manner
  - stop conditions: either reach MAX_DEPTH or current candidate pair is pruned

Effects of Suffix Filtering
- 0.9M DBLP-3GRAM, Jaccard, t=0.9

<table>
<thead>
<tr>
<th>MAX_DEPTH</th>
<th>Candidate Size</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>81,650,620</td>
<td>192.7</td>
</tr>
<tr>
<td>1</td>
<td>12,117,670</td>
<td>136.7</td>
</tr>
<tr>
<td>2</td>
<td>385,095</td>
<td>109.2</td>
</tr>
<tr>
<td>3</td>
<td>41,088</td>
<td>99.2</td>
</tr>
<tr>
<td>4</td>
<td>36,696</td>
<td>98.8</td>
</tr>
</tbody>
</table>

Experiment Settings /1
- Algorithms Compared
  - Ali-Pairs
  - PPJoin
  - PPJoin+
- Measure
  - Jaccard, Cosine
  - Candidate Size, Running Time
  - Near Duplicate Web Page Detection
    - compare with shingling

Experiment Settings /2
- Environment
  - Pentium D 3.00GHz CPU, 2GB RAM
  - Debian 4.1, GCC 4.1.2 with -O3
- Dataset

Near Duplicate Web Page Detection /1
- extract q-grams and shingles, and perform similarity join
- $r_q$ = result from TREC-32shingle
- $r_q$ = result from TREC-4gram
- Precision = $\frac{tp}{r_q}$
- Recall = $\frac{tp}{r_q}$
Near Duplicate Web Page Detection

Results:

<table>
<thead>
<tr>
<th>Threshold (Jaccard)</th>
<th>Precision</th>
<th>Recall</th>
<th>Time (shingling + ppjoin+)</th>
<th>Time (qgram-allpairs)</th>
<th>Time (qgram-ppjoin+)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.95</td>
<td>0.38</td>
<td>0.11</td>
<td>1.00s</td>
<td>41.96s</td>
<td>11.76s</td>
</tr>
<tr>
<td>0.90</td>
<td>0.48</td>
<td>0.00</td>
<td>1.03s</td>
<td>245.03s</td>
<td>43.37s</td>
</tr>
<tr>
<td>0.85</td>
<td>0.58</td>
<td>0.04</td>
<td>1.03s</td>
<td>926.54s</td>
<td>202.69s</td>
</tr>
</tbody>
</table>

100% precision & recall

Edit Similarity Join [Chuan et al, VLDB08]

- Continue to consider other similarity/distance functions
- Edit distance
  - Widely used text dissimilarity measure
  - Captures orders among characters
  - $O(n^2)$ using standard dynamic programming
- Consider similarity join with an edit distance threshold
  - i.e., find $(r, s)$ s.t. $ed(r, s) \leq d$

Prior Art: q-gram-based Method

- q-gram-based filtering [Gravano et al, VLDB01]
  - if $ed(r, s) \leq d$ at least $LB(r, s)$ common q-grams between them
  - $LB(r, s) = \max(|r|, |s|) + q + 1 - d \cdot q$
    - $| |r| - |s| | \leq d$
    - positions of the matching q-grams should be within $d$
  - Implementation via SQL & UDF
    - $q=2$ achieves best performance

Ed-Join

- Ed-Join improves the previous method
  - Location-based mismatch filtering
    - Prefix filtering with minimum prefix length (for edit distance)
  - Content-based mismatch filtering
    - Interesting experimental results
  - Idea
    - mismatching q-grams also provide useful information

Location-based Mismatch Filtering

- Prefix length = $q \cdot d + 1$ ⇒ Minimum prefix length $l \in [d+1, q \cdot d + 1]$
  - $(r, s)$ is a candidate pair only if their minimum prefixes intersect

Content-based Mismatch Filtering

- Effective for burst errors ⇐ worst case for count filtering
  - “We use Sybase” ⇒ “We use Oracle”
- $L_1$ distance within any probing window $\leq 2 \cdot d$
- $n=5, d=2$
Experimental Results
- Datasets
  - DBLP, TEXAS, TREC, UNIREF
- Algorithms
  - All-Pairs-Ed, Ed-Join, PartEnum

Optimal q-gram Length
- q-grams longer than 2 is usually (much) better!

Large Edit Errors
- ~14 sec for UNIREF (~365k * 465) with t = 20!

Top-k Similarity Join [Chuan et al, ICDE08]
- Return top-k most similar pairs of objects
- Can stop the algorithm at any time
- Get top-j most similar pairs (j ≤ k)

Incremental Similarity Join
- \( \text{sim}(x, y) = 0.9999999 \) and we want the top-1 pair, can we output (x, y)?
- Need an upper bound of unseen pairs' similarities
- 1st try: try threshold \( t \) in decreasing order, and run All-Pairs/PPJoin algorithm for \( t \)
  - High redundancy
  - May return much more results

Event-Driven Top-k Similarity Join Algorithm
- Idea: enumerate all the necessary threshold \( t \)
  - Might generate new results when x's prefix grows by 1
  - \( s_t = 1 - (r_{t-1})/|x| \)
  - Push all \( s_t \) into a max heap
- Optimizations to avoid repeated verification
  - Reduce inverted index size
  - Tighter upper bounding

\[
\text{Score} = \frac{s_t + s_{t-1}}{2}, \quad \text{where} \quad s_t = 1 - \frac{|x|}{|y|} - \frac{1}{|x|} - \frac{1}{|y|}
\]
**Roadmap**

- Motivation
- Problem Definition and Scope
- Approximate Similarity Join Algorithms
- Conclusions

**Conclusions and Future Work**

- Similarity join applications
  - Record linkage / deduplication
  - Web
  - Clustering
- We study efficient exact similarity join algorithms for strings / sets.
- Much more can/need to be done
  - Other similarity functions (e.g., in bioinformatics, multimedia databases) (e.g., similarity join in metric space)
  - Substring similarity join (e.g., for entity extraction)

**Q & A**

“The woman at the ticket counter demanded, ‘Who is John Anderson?’ Ms. Anderson recalled. She pointed at the baby stroller and said, ‘He’s right here.’ The suspect, then 2 years old, blinked his big blue eyes and happily gummed his pacifier.”

“That baby’s on the no-fly watch list,” the agent said.

Source: http://www.dvorak.org/blog/?p=6527

More Resources at [http://www.cse.unsw.edu.au/~weiw/project/simjoin.html](http://www.cse.unsw.edu.au/~weiw/project/simjoin.html)