Similarity Join Algorithms:
An Introduction

Wei Wang
University of New South Wales
Australia

http://www.cse.unsw.edu.au/~weiw
Roadmap

A. **Motivation**
B. Problem Definition and Classification
C. Similarity Join Algorithms
D. Epilogue
Objectives

- Classify existing approaches along based on several perspectives
- Explain several useful ideas in solving the problem
The Problem

- The *same* objects (to *humans*) are deemed as *different* objects by *computers*
Computers are dumb

- Numerical errors

```c
#define EPSILON 0.00001
double x;
...
if (fabs(x - 0.1) < EPSILON) {
    ...
}
```
To Err is Human

- Typo
  - Why everyone can understand this?
  - Person’s names

4. Efficient Approximate Search on String Collections (Tutorial), Marios Hadjieleftheriou and Chen Li, VLDB 2009. [PDF], [Part I], [Part II].
5. Efficient Approximate Search on String Collections (Tutorial), Marios Hadjieleftheriou, Chen Li, ICDE 2009, [PPT-Part1], [PPT-part2].
6. Quality-Aware Retrieval of Data Objects from Autonomous Sources for Web-Based Repositories, Houtan Shirani-Mehr, Chen Li, Gang Liang, Michal Shmueli-Scheuer, ICDE 2008 (poster). [PDF]
7. Communication-Efficient Query Answering with Quality Guarantees in Client-Server
Other Reasons

- Lack of consistency
  - tf-idf, tf.idf, tf*idf
- Semantically equivalent objects

A photo and its digitally modified version are bit-wise different!
Use of Similarity Functions

- The solution
  - Represent objects in a digital format
    - Typically each object represented as a set of features
  - Define a similarity function between objects
    - \( \text{sim}(x, y) \in [0, 1] \)
    - Or can define a distance function

4. Efficient Approximate Search on String Collections (Tutorial), Marios Hadjieleftheriou and Chen Li, VLDB 2009. [PDF], [Part I], [Part II].
5. Efficient Approximate Search on String Collections (Tutorial), Marios Hadjieleftheriou, Chen Li, ICDE 2009, [PPT-Part1], [PPT-part2].
6. Quality-Aware Retrieval of Data Objects from Autonomous Sources for Web-Based Repositories, Houtan Shirani-Mehr, Chen Li, Gang Liang, Michal Shmueli-Scheuer, ICDE 2008 (poster). [PDF]
7. Communication-Efficient Query Answering with Quality Guarantees in Client-Server
Example 1

Object 104-dim vector

similar? mapping

Object 104-dim vector

dist() < ε

Feature

nokia n8

Google 搜索

获得约 2,270,000 条结果（用时 0.03 秒）
Google’s Image Clustering [Liu et al, WACV07]

- Use MR + Spill Tree for kNN search in a feature space
- Image features = 104-dim real vectors
  - Normalize color intensities & picture size (to 64 x 64)
  - Extract and quantize Haar wavelet features
    - Quantize largest 60 coefficients to +/- 1
    - Others \( \Rightarrow 0 \)
  - Dimensionality reduction
    - 64 * 64 * 3-dim binary vector \( \Rightarrow \) 100-dim vector via random projection
  - Add avg color values + picture aspect ratio
Example 2

- Identify spams / plagiarism / copyright protection / replicate Web collections
  - [dejavu](http://www.dejavu.org) for MEDLINE database
  - [www.rentacoder.com](http://www.rentacoder.com)

Feature

![Diagram](attachment:feature.png)

- Extraction + tokenization / q-gram
  - Jaccard(\( \geq 0.9 \))
  - Similar?
FACT: MEDLINE DB has ~20M records

FACT: Naïve pair-wise comparison: 200 trillion ($2 \times 10^{14}$)

FACT: Take >6 years if performing 1M comparison per second
Other Applications

- Collaborative filtering
- Bioinformatics
- File/Document management systems
- Match-making services
  - Job recruitment, Dating
Roadmap

A. Motivation
B. Problem Definition and Classification
C. Approximate Similarity Join Algorithms
D. Epilogue
Problem Definition

- **Input**
  - two sets of objects: $R$ and $S$
  - a similarity function: $\text{sim}(r, s)$
  - a threshold: $t$

- **Output**
  - all pairs of objects $r \in R$, $s \in S$, such that $\text{sim}(r, s) \geq t$

- **Variations**
  - $\text{dist}(r, s) \leq d$

- If $R = S \Rightarrow$ self similarity join

- E.g., $\cos(D_i, D_j) \geq 0.9$ for near duplicate document/Web page detection.

- E.g., $\text{edit-dist}(s_i, s_j) \leq 2$ to match customers’ names.

- $t$ is usually close to 1

- $d$ is usually close to 0
Similarity/Distance Functions

- $L_p$ distance
  \[ L_p(x, y) = \left( \sum_i |x_i - y_i|^p \right)^{1/p} \]

- Hamming distance
  \[ H(x, y) = |(x - y) \cup (y - x)| \]

- `set_contains?`, `set_intersects`?

- Overlap and Jaccard
  \[ \text{contains}(x, y) = \begin{cases} 
    1 & x \subseteq y \\
    0 & x \not\subseteq y 
  \end{cases} \]
  \[ \text{overlap}(x, y) = |x \cap y| \quad J(x, y) = \frac{|x \cap y|}{|x \cup y|} \]

- Cosine similarity
  \[ \text{cosine}(x, y) = \frac{\vec{x} \cdot \vec{y}}{||x|| \cdot ||y||} \]

- Edit distance
Classification

- We look at the *ideas & techniques* used in previous work

<table>
<thead>
<tr>
<th></th>
<th>Euclidean</th>
<th>Metric</th>
<th>Others</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Exact</strong></td>
<td>Ore / MSJ / GESS</td>
<td>D-index</td>
<td>PSJ, Probe-Count-Opt, SSJoin, All-Pairs, PPJoin+, Ed-Join, Hamming distance join, PartEnum</td>
</tr>
<tr>
<td><strong>Approx.</strong></td>
<td>LSH</td>
<td></td>
<td>Shingling, simhash, l-match, SpotSigs, blocking, canopy clustering</td>
</tr>
</tbody>
</table>
Scope

- Connection to many other well-known problems
  - kNN/range search and spatial databases
  - approximate string matching
  - top-k query processing
  - dimensionality reduction (signature-based schemes)

- By no means exhaustive
  - SIGMOD06 tutorial by Koudas, Sarawgi & Srivastava
  - SAC07 tutorial by Zezula, Dohnal & Amato
  - ICDE09 tutorial by Samet
  - VLDB09 tutorial by Hadjieleftheriou & Li
  - Survey papers

- We focus on "similarity join" "algorithms"
Roadmap

A. Motivation

B. Problem Definition and Classification

C. Similarity Join Algorithms
   1. Exact algorithm
      - Euclidean
      - Metric
      - Others (set & string)
   2. Approximate algorithm

D. Epilogue
First Glance into the Problem

- **Simple variation**
  - find exact duplicate \( \Rightarrow \) \( \text{sim}(x, y) = 1 \)
  - Use hashing (e.g., SHA1, Rabin’s fingerprinting)

- **Naïve Algorithm**
  - Simple nested loop algorithm
    - Compare all \( O(n^2) \) pairs

- **Optimization opportunities \( \Rightarrow \) Be Happy!**
  - *Be lazy*: only consider promising pairs
  - *Be aggressive*: pruning-and-refinement paradigm
  - *Don’t be fussy*: Resort to approximate solutions
Challenges

- High dimensionality
  - Curse of dimensionality
  - Sparsity
- Large datasets
- Hard similarity functions
  - expensive to evaluate
  - hard to index
  - do not have nice properties (e.g., transitivity, metric)
C. Similarity Join

1. *Exact algorithm*
   - Euclidean
   - Metric
   - Others (set & string)

2. *Approximate algorithm*
Multidimensional Similarity Join

- Focus on points in a *high dimensional* Euclidean space with $L_p$ distance functions
  - $\{<r, s> | r \in R, s \in S, L_p(r, s) \leq \varepsilon\}$

- We pick Ore/MSJ/GESS as a representative method [Orenstein, SSD91] [Koudas & Sevcik, TKDE00] [Dittrich & Seeger, KDD01]

- Utilize hypercube-based filtering

![Diagram showing hypercube-based filtering and false positive cases](diagram.png)
Replication

- Only consider the finest partitions
- **Use replication if a hypercube intersects multiple partitions**
- To find overlapping hypercubes, only consider partitions \( <x, y> \), s.t.,
  - \( x = y \)
- **Problems:**
  - Too much replication
  - Need deduplication
- **Other methods**
  - \( \varepsilon \)-kdb-tree [Shim et al, ICDE 97] avoids replication but accesses neighboring partitions on-the-fly
Recursive Space Partitioning

- “Hash” hypercubes into their smallest enclosing partitions (or “buckets”)
- To find overlapping hypercubes, only consider partitions \(<x, y>\), s.t.,
  - \(x = y\)
  - or \(x\) is a prefix of \(y\)

<table>
<thead>
<tr>
<th>Partition</th>
<th>Cubes</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\phi)</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>3.3</td>
<td></td>
</tr>
</tbody>
</table>
Merge Join with a Stack

- Sort partitions based on their labels
- Perform merge join with a stack
- Generate candidate pairs when popping elements from the stack

Correct as if r overlaps s, r and s has a containment relationship

R
3  {a}
3.1 {b}
3.2 {c}

S
1.2 {d}
3.1.1 {e}
3.2 {f}

Correct as if r overlaps s, r and s has a containment relationship

stack: 1.2
3.1
3.1.1
3.2
Other Approaches

- LSS algorithm that is GPU’s parallel sort-and-search capability [Lieberman et al, ICDE08]
C. Similarity Join

1. **Exact algorithm**
   - Euclidean
   - Metric
   - Others (set & string)

2. **Approximate algorithm**
Similarity Join in Metric Space

- Three approaches experimentally studied in [Dohnal et al, ECIR 03]
  - Partition-based
    - Partition on \(d(\rho, x_i)\)
  - Filtering-based
    - Multiple pivots + triangle inequality filtering
  - Index-based
    - D-Index(\(\rho\))

- Other approaches
  - [Paredes and Reyes, SISAP 08] indexes both joining sets \textit{jointly}
  - [Jin et al, DASFAA 03] uses StringMap to derive approximate answers
C. Similarity Join

1. **Exact algorithm**
   - Euclidean
   - Metric
   - Others (set & string)

2. **Approximate algorithm**
Similarity Join for Sets & Strings

- Similarity between *sets*
  - **Binary similarity functions**
    - Contains, intersects
  - **Numerical similarity functions**
    - Overlap, Jaccard, dice, cosine

- Similarity between *strings*
  - Treat strings as sets
  - Jaccard (on q-grams), edit distance
Set Containment Join /1

- **Problem:**
  - find \( \{ (r, s) \mid r \in R, s \in S, r \subseteq s \} \)

- **PSJ Algorithm** [Ramasamy et al, VLDB00]
  - Generate candidates
    - \( \text{len+sig}(r) \rightarrow \text{hash}(\text{random-elem}(r)) \)
    - \( \text{len+sig}(s) \rightarrow \text{hash}(s[1]), \text{hash}(s[2]), \ldots \)
  - Join only corresponding partitions
    - with (length, signature) optimizations
  - **Verification**
    - Test if \( r \subseteq s \)

---

### Table

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Alice</strong></td>
<td>C, C#</td>
</tr>
<tr>
<td><strong>Bob</strong></td>
<td>C, Ruby</td>
</tr>
<tr>
<td><strong>Charlie</strong></td>
<td>Lisp, Haskell</td>
</tr>
<tr>
<td><strong>MS</strong></td>
<td>C, C#, VB</td>
</tr>
<tr>
<td><strong>Sun</strong></td>
<td>Java</td>
</tr>
<tr>
<td><strong>Google</strong></td>
<td>C, C++, Python</td>
</tr>
</tbody>
</table>
Set Containment Join /2

- Other methods
  - signature hash join [Helmer & Moerkotte, VLDB97]
  - index-nested-loop-join is faster, even building an in-memory index on-the-fly [Mamoulis, SIGMOD03]
Similarity Join for Sets & Strings

• Similarity between sets
  • Binary similarity functions
    • Contains, intersects
  • Numerical similarity functions
    • Overlap, Jaccard, dice, cosine

• Similarity between strings
  • Treat strings as sets
  • Jaccard (on q-grams), edit distance
Set Similarity Join

- **Problem**
  - find \( \{(r, s) \mid r \in R, s \in S, \text{overlap}(r, s) \geq t\} \)
  - self-join: \( \{(s_i, s_j) \mid s_i \in R, s_j \in S, i \leq j, \text{overlap}(s_i, s_j) \geq t\} \)
  - A fundamental “operator”
    - can handle other similarity functions (Jaccard, cosine, Hamming, dice, edit distance, …) via transformation

- **Optimization opportunities:**
  - Typically \( t \) is very close to 1.0
  - The distribution of elements among the records are not uniform
Framework

- for each $S_j \in S$
  - Candidates = $\phi$
  - Candidates = getCandidates($S_j$)
  - Verify($S_j$, Candidates)

- Algorithms differ in getCandidates()
  - naïve algorithm:
    - return $\{S_i \mid i < j\}$
  - naïve inverted-index-based algorithm:
    - return $\{S_i \mid i < j \land S_i \cap S_j \neq \phi\}$ // use inverted index

```
Verify(x, {y_1, y_2, ...})
for each y_i
  if overlap(x, y_i) \geq t
    output(<x, y_i>)
end
```
Probe-Count /1

- Probe-Count-Opt Algorithm [Sarawagi & Kirpal, SIGMOD04]
  - index-nested-loop-join style
  - for each tuple, invoke an optimized version of list merge with threshold algorithm
**Probe-Count /2**

- Upper bounding the overlap

-- overlap constraint: \( t = 3 \)

-- current record \( S_j = \{a, b, c, d, e\} \)

<table>
<thead>
<tr>
<th>a</th>
<th>1 3 5 7 9 11 13 15</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>2 4 6 8 9 12 14</td>
</tr>
<tr>
<td>c</td>
<td>1 5 7</td>
</tr>
<tr>
<td>d</td>
<td>3 9</td>
</tr>
<tr>
<td>e</td>
<td>2</td>
</tr>
<tr>
<td>f</td>
<td>1 9 15 27</td>
</tr>
</tbody>
</table>

\[ t-1 = \{1, 2, 3, 5, 7, 9\} \]

Verify 1: \( \text{binary_search}(l(b), 1) = \text{false} \)
\[ \Rightarrow \text{overlap}(\text{cur}, 1) \leq 2 \]

Verify 2: …
Probe Count /3

- Other optimizations
  - Sorting by increasing record size
  - Clustering
  - External memory version

- Other methods
  - ScanCount, MergeSkip, Divide-Skip [Li et al, ICDE08]

- Comment on Probe-Count-Opt
  - Only the rarest $|S| - (t-1)$ tokens are used to generate candidates
  - Verification may be quite expensive
  - Unnecessary candidates generated (and verified)

\[ e.g., t = 0.9 \times |S| \]
Prefix Filtering-based Similarity Joins

- **SSJoin** [Chaudhuri et al, ICDE06]
  - Formalize the *prefix-filtering* principle and use it in a *symmetric* way
  - Access original records for verification

- **All-Pairs** [Bayardo et al, WWW07]
  - Use prefix-filtering in an *asymmetric* way

- **PPJoin+** [Xiao et al, WWW08]
  - Employs prefix-filtering, *position filtering* and *suffix filtering*
Prefix Filtering /1

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

What's the maximum possible number of cards held by both players (denomination not considered)?
Prefix Filtering /2

Formally

- $\text{Prefix}_t(U) \cap \text{Prefix}_t(V) = \emptyset \implies \text{overlap}(U, V) < t$
- i.e., $(U, V)$ can be safely pruned
- Global ordering important
  - Use globally rare tokens in the prefixes

prefix-len = $|U| - (t-1)$ for overlap similarity function
**Framework**

- for each $S_j \in S$
  - $\text{Candidates} = \emptyset$
  - $\text{Candidates} = \text{getCandidates}(S_i)$
  - $\text{Verify}(S_j, \text{Candidates})$

- Algorithms differ in $\text{getCandidates}()$
  - naïve alg: return $\{S_i \mid i < j\}$
  - index-based alg: return $\{S_i \mid i < j \land S_i \cap S_j \neq \emptyset\}$ // use inverted index
  - prefix-filtering-base alg: return $\{S_i \mid i < j \land \text{prefix}(S_i) \cap \text{prefix}(S_j) \neq \emptyset\}$
All-Pairs [Bayardo et al, WWW07]

- All-Pairs tackles (weighted) cosine and Jaccard similarity functions

- It also improves SSJoin
  - stand-alone implementation
  - tight transformation between similarity/distance functions
  - hash table instead heap
  - indexing & probing prefixes
Relationships among Similarity/Distance Functions

- **Jaccard similarity**
  - \( J(x, y) = \frac{|x \cap y|}{|x \cup y|} \)
  - \( J(x, y) \geq t \iff O(x, y) \geq \frac{t}{1+t} \cdot (|x| + |y|) \)
  - \( J(x, y) \geq t \Rightarrow |y| \geq t \cdot |x| \) (wlog. if \(|y| \leq |x|\))

- **Cosine similarity**
  - similar transformations can be obtained.
  - \( \cos(x, y) \geq t \)
  - \( J(x, y) \geq t \Rightarrow |y| \geq t^2 \cdot |x| \) (wlog. if \(|y| \leq |x|\))

- **Edit distance**
Prefix Lengths

- **Jaccard similarity**
  - \( J(x, y) \geq t \iff O(x, y) \geq \frac{t(1+t)}{1+t} \cdot (|x| + |y|) \)
  - \textit{indexing}-prefix-len = \( |x| - \left\lfloor \frac{2t}{1+t} \cdot |x| \right\rfloor + 1 \)
  - \textit{probing}-prefix-len = \( |x| - \left\lfloor t \cdot |x| \right\rfloor + 1 \)

- **Cosine similarity**
  - \( \cos(x, y) \geq t \iff O(x, y) \geq t \cdot (|x| \cdot |y|)^{1/2} \)
  - \textit{indexing}-prefix-len = \( |x| - \left\lfloor t \cdot |x| \right\rfloor + 1 \)
  - \textit{probing}-prefix-len = \( |x| - \left\lfloor t^2 \cdot |x| \right\rfloor + 1 \)

[Xiao et al, WWW08]
[Bayardo et al, WWW07]
All-Pairs

- for each $S_j \in S$ in increasing size  
  - Candidates = $\emptyset$
  - prefix-len = $\text{calc}_\text{probing}_\text{prefix}_\text{len}()$
  - for i=1 to prefix-len  
    - $w = S_j[i]$
    - for each $S_k \in \text{Inverted-list}(w) \& \text{len-filter}$  
      - Candidates = Candidates $\cup$ $S_k$
      - If i < $\text{calc}_\text{indexing}_\text{prefix}_\text{len}()$
        - Inverted-list(w) = Inverted-list(w) $\cup$ $S_j$
    - $\text{Verify}(S_j, \text{Candidates})$

\begin{tabular}{|c|c|}
\hline
$\text{Verify}(x, \{y_1, y_2, \ldots\})$  
\hline
for each $y_i$  
  if overlap(x, $y_i$) $\geq t$  
  output($<x, y_i>$)  
\hline
\end{tabular}
<table>
<thead>
<tr>
<th>RID</th>
<th>Name</th>
<th>len</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Database System Concepts</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>Database Concepts Techniques</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>Database System Programming Concepts Oracle Linux</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>Database Programming Concepts Illustrated</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>System Programming Concepts Techniques Oracle Linux</td>
<td>6</td>
</tr>
</tbody>
</table>

**Order:** Illustrated, Linux, Oracle, Techniques, Programming, System, Database, Concepts

<table>
<thead>
<tr>
<th>token</th>
<th>df</th>
<th>Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>System</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>Concepts</td>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td>Techniques</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>Programming</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>Oracle</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Linux</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Illustrated</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>RID</td>
<td>Name</td>
<td>len</td>
</tr>
<tr>
<td>-----</td>
<td>--------------------------------</td>
<td>-----</td>
</tr>
<tr>
<td>1</td>
<td>System Database Concepts</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>Techniques Database Concepts</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>Illustrated Programming Database Concepts</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>Linux Oracle Programming System Database Concepts</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>Linux Oracle Techniques Programming System Concepts</td>
<td>6</td>
</tr>
</tbody>
</table>

### length filtering does not help in this toy example

<table>
<thead>
<tr>
<th>token</th>
<th>Inverted list</th>
</tr>
</thead>
<tbody>
<tr>
<td>System</td>
<td>{1}</td>
</tr>
<tr>
<td>Techniques</td>
<td>{2}</td>
</tr>
<tr>
<td>Illustrated</td>
<td>{3}</td>
</tr>
<tr>
<td>Linux</td>
<td>{4, 5}</td>
</tr>
</tbody>
</table>

cur RID = 12:4 5

Cand={4}
**PPJoin+ [Xiao et al, WWW08]**

- PPJoin+ improves All-Pairs
  - Optimized for Jaccard/cosine similarity constraints
  - less candidates generated
  - less full-scale verifications
- Idea: *fully* exploit the global ordering
  - Record the position of the tokens in the prefix $\Rightarrow$ ppjoin
  - Probe the tokens in the suffixes $\Rightarrow$ ppjoin+
How Positional Information Helps

- Derive an upper bound of the overlap based on position information in the prefixes

\[ \text{overlap}(x, y) \leq 1 + \min\{ 5-2, 5-1 \} = 4 \leq \alpha = 5 \]

\( \Rightarrow \langle x, y \rangle \) is NOT a candidate pair

prefix(x) \( \cap \) prefix(y) \( \neq \) \( \phi \)

\( \Rightarrow \langle x, y \rangle \) is a candidate pair
Suffix Filtering

- Can position information be used to the suffixes?

- \( \langle x, y \rangle \) is not a candidate pair for \( t=0.8 \)
  - Overlap\((x, y)\) must be \( \geq 16 \)

Apply multiple probes in a divide-and-conquer manner
Edit Similarity Join

- **Edit distance**
  - Widely used text dissimilarity measure
    - Models human errors (e.g., typos)
  - Expensive to evaluate
    - $O(len^2)$ using standard dynamic programming

- **Similarity join with an edit distance threshold**
  - i.e., find $(r, s)$ s.t. $ed(r, s) \leq d$
  - Seems the only way is to perform pair-wise comparisons
q-gram-based Method

- q-gram-based filtering
  [Gravano et al, VLDB01]
  - if \( \text{ed}(r, s) \leq d \) \( \Rightarrow \)
    \[
    \text{overlap}(r^*, s^*) \geq \max(|r|, |s|) + q + 1 - d^*q
    \]
  - \( | |r| - |s| | \leq d \)
  - positions of the matching q-grams should be within \( d \)

- Implementation via SQL & UDF
  - \( q=2 \) achieves best performance

Implication: Edit similarity join can be processed using other similarity join algorithm
Ed-Join  [Xiao et al, VLDB08]

- 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** intersects

\( q = 2, \ d = 1 \)

```
abaabab  \rightarrow  (aa, 3) (ab, 1) (ab, 4) (ab, 6) (ba, 2) (ba, 5)  \text{min-prefix}

xx yabab  \rightarrow  (xx, 1) (xy, 2) (ab, 4)  \text{min-prefix}
```

- \( \rightarrow \) less candidates

- **Count filtering** is a special case of **location-based mismatch filtering**
Content-based Mismatch Filtering

- Effective for burst errors $\leftarrow$ worst case for count filtering
  - “We use Sybase” $\rightarrow$ “We use Oracle”
- $L_1$ distance within any probing window $\leq 2*d$

$q=5, d = 2$

$C_1 C_2 C_3 C_4 C_5 C_6 C_7 C_8 C_9 C_{10} \ldots C_{25}$

$C_1 C_2 C_3 C_4 Z_1 Z_2 Z_3 Z_4 Z_5 C_{10} \ldots C_{25}$

Probing Window
Optimal q-gram Length

q-grams longer than 2 is usually (much) better!
Other Approaches

- **Variants of q-gram**
  - VGRAM [Li et al, VLDB07], proposes variable-length q-grams
  - Gapped q-gram [Burkhardt & Kärkkäinen, CPM02], only applicable to $d=1$

- **Neighborhood generation**
  - FastSS [Bocek et al, ETH TR 07] use deletions only and achieve $O(d \times \sum^k \times \log(n\sum^k))$ similarity query time and $O(n\sum^k)$ space.
  - [Wang et al, SIGMOD09], reduces the complexity to $O(c \times d^2)$

- **Divide-and-conquer based on pigeon hole principle**
  - Hamming Distance Join [Manku et al, WWW07]
  - PartEnum [Arasu et al, VLDB06]

Some novel methods are being developed …
Partitioning-based Approaches

- **Enumeration + Divide and conquer**
  - Hamming Distance Join [Manku et al, WWW07]
  - PartEnum [Arasu et al, VLDB06]
  - both works for Hamming distance threshold, but other constraints can be easily transformed to Hamming distance constraint, e.g.,

\[
J(x, y) \geq t \iff H(x, y) \leq \frac{1 - t}{1 + t} \cdot (|x| + |y|)
\]
Hamming Distance Join [Manku et al, WWW07]

- **Background**
  - N docs mapped to sketches of f-bits each (using simhash [Charikar, STOC02])
  - given a new document, generate its sketch q
  - need to return all sketches that has Hamming distance at most t from q, i.e., \( \text{Hamming}(x, y) \leq t \)

- No “good” theoretical solutions
- Naïve solutions
  - Query expansion | OR | Data replication
    - too many queries | this proposal | too many copies
Hamming Distance Query [Manku et al, WWW07]

- if v is an answer, v and q differ by at most t bits
  - but these t bits can be anywhere within the [1 .. f]

**Solution:** partition

<table>
<thead>
<tr>
<th>q</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
</table>

| v_1 | 1 | 1 | 0 | 1 | ? | ? |
| v_2 | 1 | 1 | 0 | 1 | ? | ? |

- Duplicate data 3 times (or index)
- \( E(|C_i|) = \frac{N}{2^t} \)
- Design of parameters important

Elements in \( C_i \) need further verification

\[ \begin{array}{ccccccc}
? & ? & ? & ? & 0 & 0 \\
\end{array} \]

\( C_1 = \{a, \ldots\} \)
\( C_2 = \{x, \ldots\} \)
\( C_3 = \{m, \ldots\} \)

Problem:
- \#Duplication = \( C(n, t) \)
- \( |\text{Cand}_i| = \frac{N}{2^{(n-t)c}} \)
- Cannot deal with large t

form n=3 partitions

How many partitions are preserved?
**PartEnum** [Arasu et al, VLDB06]

- **Part** + **Part** + **Enum**

  - **Part** (n1=2 partitions)
    - Each record generates $n_1 \left( \frac{n_2}{k/n_1} \right)$ signatures
    - $\text{Hamming}(u, v) \leq k \Rightarrow \text{sigs}(u) \cap \text{sigs}(v) \neq \phi$

  - **Enum** (n2=3 partitions)
    - $t = 10$
      - ENUM with $n=12 \Rightarrow 66$ sigs / record
      - PartEnum with $n_1=3$, $n_2=4 \Rightarrow 12$ sigs / record

At least one partition has error $\leq \lceil t/2 \rceil = 1$ ← Pigeon hole principle

**form 2 partitions**

$$
\begin{array}{c}
\text{v} \\
1 & 1 & 0 & 1 & 0 & 0
\end{array}
$$

**form 3 2nd-level partitions**

$$
\begin{array}{c}
1 & 1 & 0 \\
1 & 0 & 0
\end{array}
\Rightarrow \{ ?10, ?01, 11? \}
$$

$$
\begin{array}{c}
1 & 0 & 0
\end{array}
\Rightarrow \{ ?00, ?10, 10? \}
$$
C. Similarity Join

1. Exact algorithm
   - Euclidean
   - Metric
   - Others (set & string)

2. Approximate algorithm
Approximate Similarity Join Algorithms

- Sketch-based methods (for metric space)
  - LSH
  - Shingling
  - Randomized Projection
  - Theoretical Guarantee on the Approximation
  - Still hard to perform the join on the sketches

- Heuristic methods
  - Blocking, Canopy
  - I-Match [Chowdhury et al, TOIS 02, Kolcz et al, SIGKDD 04]
  - SpotSigs
  - Works well for specific types of applications/datasets
The Idea

- X and Y are very similar $\Rightarrow$ partOf(X) is “very very similar” to partOf(Y)

Diagram:
- Very very Similar
- Identical
- Projection
- Sorted Neighborhood
- Fuzzy blocking
- LSH
- Token / q-gram
- I-Match
- Super-shingling
- Canopy
- Canopy blocking
- Fuzzy blocking
- I-Match
- Super-shingling
- Canopy
Locality Sensitive Hashing

- LSH solves nearest neighbor problem approximately \cite{IndykMotwaniSTOC98, GionisVLDB99, IndykFOCS00} ... \cite{AndoniIndykFOCS06}
  - Widely used, e.g., multimedia database & computer vision

- Idea:
  - encourage collision of $h(x)$ and $h(y)$ when $x \approx y$
  - contrast this with traditional & cryptology hash functions
Definition

- LSH: a family $H$ is called $(R, cR, p_1, p_2)$-sensitive if for any two points $x, y \in \mathbb{R}^d$
  - if $d(x, y) \leq R \implies \Pr_H[h(x) = h(y)] \geq p_1$
  - if $d(x, y) \geq cR \implies \Pr_H[h(x) = h(y)] \leq p_2$

$c > 1$
$p_1 > p_2$
**LSH Cookbook**

- **Known LSH families**
  - $\mathbb{R}^d$, Hamming distance
    - $h_k(x) = x_k$, i.e., random projection on one dimension
  - $\mathbb{R}^d$, $L_1$ distance
  - $\mathbb{R}^d$, $L_p$ distance
    - $h_{r,b} = [(r \cdot x + b) / w]$, $r[i]$ is sampled from Gaussian distribution
    - $p$-stable distribution for $p \in [0, 2]$
  - Jaccard: *min-hashing*
  - arccos: *simhash*
  - $L_2$ distance on a unit hypersphere [Terasawa & Tanaka, WADS07]
Shingling

- Doc D $\rightarrow$ set of Shingles (aka. q-grams)
  - $\text{Sim}(D_i, D_j) = \text{Jaccard}(\text{Shingles}(D_i), \text{Shingles}(D_j))$

- Consider the universe $U = |R \cup S|$
  - Random (wrt U) sample one element from R and S
  - $P[\text{sample}(R) = \text{sample}(S)] = |R \cap S| / |R \cup S| = \text{Jaccard}$

- However, we don’t know $U$ beforehand
  - Min-hashing
    - randomly (wrt I) permutate $e_i \in R$ $\rightarrow$ hash($e_i$)
    - select the first element after permutation $\rightarrow$ $\text{sig}(R) = \min_i \{ \text{hash}(e_i) \}$
Shingling Example

- \( \text{Jaccard}(R, S) \approx \frac{\text{COUNT}( h(R) = h(S) )}{N} \)

\[
\begin{align*}
\text{hash}_1 &= 2x \mod 5 \\
\text{hash}_2 &= 3x \mod 7
\end{align*}
\]
Joining the Signatures

- Doc $D \rightarrow$ set of Shingles (aka. q-grams)
  - $\text{Sim}(D_i, D_j) = \text{Jaccard}(\text{Shingles}(D_i), \text{Shingles}(D_j))$

- Doc $D \rightarrow$ set of signatures (of shingles)
  - $\text{Sim}(D_i, D_j) = \text{Overlap}(\text{sig}(s)(D_i), \text{sig}(s)(D_j)) / N$

- Still expensive for exact join
  - Remove frequent shingles [Heintze 1996]
  - Retain only every 25$^{th}$ shingle [Broder et al, WWW97]
  - with both optimizations, 10 days for 30M docs
  - Super-shingling, with overlap threshold = 1
SimHash

- Generalization of LSH to other similarity measures
  [Charikar, STOC 02]

  - $\theta(x, y)$: related to cosine
  - $h_u(x) = sign(u \cdot x)$, where $u$ is a random unit vector
  - then $\Pr[h_u(x) = h_u(y)] = 1 - \frac{\theta(x, y)}{\pi}$
Practical Implementation

- Near duplicate Web page detection from google [Henzinger, SIGIR06] [Manku et al, WWW07]
  - Document D \(\rightarrow\) set of tokens with idf weighting \(\rightarrow\) form a set of “features” \(v(D)\)
  - Each feature is randomly projected to \(f\)-dimensional binary vector of \([-1,1]\)
  - Sum up the weighted projections of all features in \(v(D)\) \(\rightarrow r(D)\)
  - a \(f\)-bit signature \(\text{sig}(D) \leftarrow \text{sign}(r(D))\)

- Results (in comparison with Shingling)
  - Fairly accurate and stable
  - Does not capture order among tokens
SimHash Example
Approximate Join Algorithms Without Quality Guarantees

- Application areas:
  - Record linkage, data cleaning
  - Clustering

- Algorithms:
  - Standard blocking
  - Sorted neighborhood
  - Fuzzy blocking
  - Canopy clustering
Blocking

- Standard blocking [Jaro, JASS89]
- Idea: similar records usually have *identical* feature values
- Algorithm:
  - GROUP BY the blocking key (e.g., `lastname[1..4]`)
  - pair-wise comparison within each group
- Limitations
  - Strong assumption (e.g., no typo in `lastname[1..4]`)
  - Recall depends on the choice of the blocking key
Sorted Neighborhood
[Hernandez & Stolfo, SIGMOD95]

- **Application:**
  - merging records from multiple sources, using *complex* similarity functions
- **Idea:** similar records usually have *similar* feature values
- **Algorithm:**
  - create a key for every record (e.g., `lastname[1..4]`)
  - sort data wrt the key
  - pair-wise comparison within a sliding window of size `w`
- **Moral:** Multi-pass + transitive closure > single-pass (large `w`)
- **Limitations:** only allow limited errors on the key

\[ ed(x.f\text{name}, y.f\text{name}) < 3 \land \land geo-dist(x.addr, y.addr) \Rightarrow x = y \]
Fuzzy Blocking

- **Bigram Indexing** [Christen & Churches, Febrl, 2003]
- Allow small errors in the key by
  - requiring only a fraction of bigrams are preserved
  - insert the record into multiple blocks
- E.g., key value = “abcde”, and we require 70% bigrams preserved
  - Generate all 4 possible combinations, insert into corresponding blocks
  - e.g., {ab, bc, cd}, {ab, bc, de}, {ab, cd, de}, {bc, cd, de}
Canopy Clustering

- Canopy Clustering as a solution to tackle *hard* clustering problems [McCallumzy et al, KDD00] [Cohen & Richman, KDD02]
  - millions of points
  - many thousands of dimensions
  - many thousands of clusters

- Idea
  - Create overlapping canopies (i.e., special subsets)
  - Perform clustering but do not consider \((x, y)\) if they never appear in one canopy
I-Match Algorithm

1. Doc $\rightarrow$ Bag of tokens $\rightarrow$ Sorted set of unique tokens $\rightarrow$ Prune tokens wrt idf values $\rightarrow$ SHA digest
   - “Hello World and Hello Web” $\rightarrow$ … $\rightarrow$ [and, Hello, Web, World] $\rightarrow$ [Hello, Web, World] $\rightarrow$ 0x685b…..

2. $[d_1, SHA_1], [d_2, SHA_2], …$
   - collision on SHA digest values $\rightarrow$ near duplicate document

- 18K Web docs $\rightarrow$ 83 sec (I-Match) vs ~590 sec (Shingling)
- It is shown that pruning tokens s.t. $\text{nidf(token)} < 0.1$ results in most accurate results for near-duplicate detection
  - effectively, ignoring frequently occurring tokens
SpotSigs [Jonathan et al, SIGIR07]

- Frequently occurring tokens are useful
  - Serve as anchors
  - Closely related to document fingerprinting methods

SpotSigs

- Choose set of <antecedent, spot dist>
  - e.g., <“are”, 2>, <“to”, 3>
  - Sig(till_here) = { “are” → “serve”, “to” → “methods”, “to” → “till_here” }
  - \( \text{sim}(X \Rightarrow Y) = \frac{|\text{sig}(X) \cap \text{sig}(X)|}{|\text{sig}(X)|} \)
Roadmap

A. Motivation
B. Problem Definition and Scope
C. Similarity Join Algorithms
D. Epilogue
   1. Recurring ideas
   2. Open issues
Recurring Ideas /1

- Similar objects should be also similar in some feature space
  - MBRs in R-tree
  - M-tree
  - Randomized projection
  - Canopy (distance wrt a pivot)

- Replication
  - Replication in spatial join
  - Neighborhood generation
  - Hamming sim join, PartEnum
Recurring Ideas /2

- Index
  - Set containment join
  - All-Pairs, PPJoin+, Ed-Join

- Pruning
  - Derive lower/upper-bounding techniques to prune candidates as early as possible

- Partitioning
  - Length partition in All-Pairs, PartEnum
  - Reduce approximate distance threshold by Pigeon-hole principle
Open Issues /1

- Further optimization on performances
  - Index for similarity functions (e.g., cosine)
  - Better pruning techniques
  - Optimize for the specific similarity/distance function
Open Issues /2

- To the base of the iceberg
  - Color histogram intersection, earth moving distance in multimedia databases
    \[ \sum_{i} \min(x[i], y[i]) / \min(\sum_{i} x[i], \sum_{j} y[j]) \]
  
  - Dynamic time warping in speech recognition and time-series databases
    \[ D(i, j) = d(i, j) + \min(D(i - 1, j), D(i, j - 1), D(i - 1, j - 1)) \]

- Similarity functions for data integration / record linkage
  \[ d_{j}(x, y) = \frac{1}{3} \left( \frac{m}{|x|} + \frac{m}{|y|} + \frac{m - t}{m} \right) \]
  \[ d_{jw}(x, y) = d_{j}(x, y) + l \cdot p \cdot (1 - d_{j}(x, y)) \]
Open Issues /2

• To the base of the iceberg
  • *Similarity functions for protein sequences*
    • Smith & Waterman local alignment vs. BLAST

• *Tree edit distance* (Similarity between XML or Web documents)
  • [Yang et al, SIGMOD05]

• *Graph distance* (isomorphism, maximal common subgraph, …)
Open Issues /3

• New trends
  • Black-box style similarity function
    • e.g., from the output of a classifier [Chandel et al, SIGMOD07]
  • Learning and using (domain-specific) transformation rules [Arasu et al, VLDB09]
  • Query optimization problem
    • e.g., combination of multiple similarity functions [Chaudhuri et al, VLDB07]
Objectives Revisited

- Classify existing approaches along based on several perspectives
  - Euclidean space / metric space / other
  - Exact / approximate

- Explain several useful ideas in solving the problem
  - Partitioning
  - Lower/upper bounding
  - Similarity function specific filtering
  - Synopsis / signature
Q & A

Our Similarity Join Project Homepage:
http://www.cse.unsw.edu.au/~weiw/project/simjoin.html