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
Computers are dumb

C:\Users\weiw>python
ActivePython 2.5.1.1 (ActiveState Software Inc.) based on ...
>>> 1.23 / 2.0
0.61499999999999999

- Numerical errors

```c
double x;
...
if (fabs(x - 0.1) < EPSILON) {
    ...
}
```
Humans are incomprehensible

- Typo
  - Why everybody can understand this?
- Lack of consistency

炜煒

- Lack of precision
  - a photo and its digitally-modified version are bit-wise different!
Notion of Similarities /1

similar !

mapping

sim() > t

Object → Feature Values
Object → Feature Values
Notion of Similarities /2

SIFT (+ PCA)

similar!

Object → Feature Values

Object → Feature Values

$L_2() \leq t \& RANSAC$
On one end, a winded Pete Sampras tried to summon enough energy to give the New York fans another memorable win to talk about it 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 he wants.
App: Deduplication /2

- Identify spams / plagiarism / copyright protection / replicate Web collections
  - dejavu for MEDLINE database
  - www.rentacoder.com

extraction + tokenization / q-gram

Doc → Tokens

Doc → Tokens

Jaccard() >= .9

similar!
App: Data Integration / Record Linkage

- Merge databases
  - TEXAS
    - burton_ty_r_3412_provine_road_mc_kinney_tx75070085103108081407000000000155109000020000006402
    - burton_ty_r_3412_provine_rd_mc_kinney_tx750700851031080814070000000155109001020000006402
  - DBLP
    - evanthia_papadopoulou_i
    - evanthia_papadopoulou_l

\[
\text{String} \xrightarrow{\text{char}} \text{Tokens}
\]

\[
\text{edit-distance()} \leq 2
\]
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**
  - $\text{dist}(r, s) \leq d$

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?**
  \[ contains(x, y) = \begin{cases} 1 & , x \subseteq y \\ 0 & , x \not\subseteq y \end{cases} \]

- **Overlap and Jaccard**
  \[ overlap(x, y) = |x \cap y| \quad J(x, y) = \frac{|x \cap y|}{|x \cup y|} \]

- **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
  - 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 \( \Rightarrow sim(x, y) = 1 \)
  - 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

- $d(r, s) > \varepsilon$ for false positive candidates
- $d(r, s) \leq \varepsilon$ for genuine candidates
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>![Image of cube]</td>
</tr>
<tr>
<td>3</td>
<td>![Image of cube]</td>
</tr>
<tr>
<td>3.3</td>
<td>![Image of cubes]</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

<table>
<thead>
<tr>
<th>( R )</th>
<th>( S )</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>{1.2}</td>
</tr>
<tr>
<td>3.1</td>
<td>{3.1.1}</td>
</tr>
<tr>
<td>3.2</td>
<td>{3.2}</td>
</tr>
</tbody>
</table>

Correct as if \( r \) overlaps \( s \), \( r \) and \( s \) has a containment relationship
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 \( d(p, x_i) \)
  - Filtering-based
    - Multiple pivots + triangle inequality filtering
  - Index-based
    - D-Index(\( \rho \))

- Other approaches
  - [Paredes and Reyes, SISAP 08] indexes both joining sets jointly
  - [Jin et al, DASFAA 03] uses StringMap to derive approximate answers

\[
d(y, z) \geq d(x, z) - d(x, y)
\]
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
    - \( \text{len} + \text{sig}(r) \rightarrow \text{hash}(\text{random-elem}(r)) \)
    - \( \text{len} + \text{sig}(s) \rightarrow \text{hash}(s[1]), \text{hash}(s[2]), \ldots \)
  - Join only corresponding partitions
    - with (length, signature) optimizations
  - Verification
    - Test if \( r \subseteq s \)
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]
Set Similarity Join

- **Problem**
  - find \( \{(r, s) \mid r \in R, s \in S, \text{overlap}(r, s) \geq t\} \)
  - A fundamental “operator”
    - can handle other similarity functions (Jaccard, cosine, Hamming, dice, edit distance, ...) via transformation

- **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 /1

- Upper bounding the overlap

-- overlap constraint: t = 3
-- current record = \{a, b, c, d, e\}

\[
\begin{align*}
a & : 1 \ 3 \ 5 \ 7 \ 9 \ 11 \ 13 \ 15 \\
b & : 2 \ 4 \ 6 \ 8 \ 9 \ 12 \ 14 \\
c & : 1 \ 5 \ 7 \ 19 \\
d & : 3 \ 4 \ 9 \ 14 \\
e & : 2 \ 8 \ 9 \ 11 \\
f & : 1 \ 9 \ 15 \ 27
\end{align*}
\]

Candidates = \( I(a) \cup I(b) \cup I(c) \cup I(d) \cup I(e) \)

\[
\begin{align*}
\text{Candidates} & = I(c) \cup I(d) \cup I(e) \\
& = \{1, 2, 3, 4, 5, 7, 8, 9*2, 11, \ldots\}
\end{align*}
\]

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

Verify 2: …
Probe Count /2

- 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)

\[ e.g., t = .9 \times |S| \]
Prefix Filtering-based Similarity Joins

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

\[
\text{if } t=4, \quad \text{overlap}(\text{player1}, \text{player2}) < t \quad \text{or} \\
\text{upperbound}(\text{overlap}(p1,p2))=t-1
\]

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

- Formally
  - $\text{Prefix}_t(U) \cap \text{Prefix}_t(V) = \emptyset \Rightarrow \text{overlap}(U, V) < t$
  - i.e., $(U, V)$ can be safely pruned
  - Global ordering important

- Algorithm (on top of an RDBMS)
  - Compute prefix$(S)$ for each record $S$
  - Candidates = $\{ \text{pairs of records that share at least one token in their prefixes} \}$
  - Verify(Candidates)

Prefix-len = $|U| - (t-1)$ for overlap similarity function
All-Pairs [Bayardo et al, WWW07]

- All-Pairs improves SSJoin
  - stand-alone implementation
  - tight transformation between similarity/distance functions
  - hash table instead heap
  - indexing & probing prefixes

- also tackles weighted cosine similarity join
**Relationships among Similarity/Distance Functions**

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

- **Cosine similarity**
  - similar transformations can be obtained.
  - \( \cos(x, y) \geq t \iff O(x, y) \geq t \sqrt{|x \cdot y|} \)
  - \( J(x, y) \geq t \Rightarrow |y| \geq t^2 \cdot |x| \) \hspace{1cm} (wlog. if \( |y| \leq |x| \))

- **Edit distance**
All-Pairs

• for each $S_j \in S$ in increasing size // nested loop
  • Candidates = $\phi$
  • prefix-len = $\text{calc\_probing\_prefix\_len}()$
  • for i=1 to prefix-len // go thru probing prefix
    • $w = S_j[i]$
      • for each $S_k \in \text{Inverted\_list}(w) \& \text{len\_filter}$ // prefix($S_k$) and prefix($S_j$)
        • Candidates = Candidates $\cup S_k$ // intersects
      • If $i < \text{calc\_indexing\_prefix\_len}()$
        • Inverted-list($w$) = Inverted-list($w$) $\cup S_j$ // index the current token
    • Verify($S_j$, Candidates)
Prefix Lengths

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

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

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

<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</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>Techniques</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>Illustrated</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>Linux Oracle</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>Linux Oracle</td>
<td>6</td>
</tr>
</tbody>
</table>

Cur RID = 1 2 4 5

<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>

Length filtering does not help in this toy example

Jaccard, t=0.8
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 → ppjoin
  - Probe the tokens in the suffixes → ppjoin+
How Positional Information Helps/1

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

\[
\text{pos} \begin{array}{cc}
1 & 2 \\
\end{array}
\]

\[
\begin{array}{cc}
x
A & B & C & D & E \\
\end{array} \rightarrow \begin{array}{cc}
A & B \\
\end{array}
\]

\[
\begin{array}{cc}
y
B & C & D & E & F \\
\end{array} \rightarrow \begin{array}{cc}
B & C \\
\end{array}
\]

**PPJoin**

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

\[\Rightarrow \langle x, y \rangle \text{ is NOT a candidate pair}\]

**All-Pairs**

\[
\text{prefix}(x) \cap \text{prefix}(y) \neq \emptyset
\]

\[\Rightarrow \langle x, y \rangle \text{ is a candidate pair}\]
How Positional Information Helps/2

- Also useful in verification

\[
\text{overlap}(x, y) \leq 1 + 4
\]
Can position information be used to the suffixes?

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

\[ \text{prefix} \quad \text{suffix} \]

\[
\begin{array}{cccc}
A & B & D & E \\
\end{array}
\]

\[
\begin{array}{cccc}
A & C & D & E \\
\end{array}
\]

\[
\begin{array}{c}
Q \\
\end{array}
\]

Overlap\((x, y)\) \( \leq 15 \)
**ppjoin+ /2**

- Apply multiple probes in a divide-and-conquer manner

  - stop conditions: either reach MAXDEPTH or current candidate pair is pruned

```
prefix | suffix
A       | B
B       | C
C       | D
```

\[
\text{ubound}_{\text{dep}=1} = 4 + 6 + 1 + 7 = 18
\]

\[
\text{ubound}_{\text{dep}=2} = 4 + 3 + 1 + 1 + 3 + 1 + 3 = 17
\]

\[
\text{ubound}_{\text{dep}=3} = 4 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 = 15
\]
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. $ed(r, s) \leq d$
**q-gram-based Method**

- q-gram-based filtering
  
  [Gravano et al, VLDB01]
  
  - if $ed(r, s) \leq d$ \(\Rightarrow\) at least $LB(r,s)$ common q-grams between them
  - $LB(r, s) = \max(|r|, |s|) + q + 1 - d^*q$

  - $| |r| - |s| | \leq d$
  - 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*d + 1 ⇒ *Minimum* prefix length \( l \in [d+1, q*d+1] \)
  - (r, s) is a candidate pair *only if* their *minimum prefixes* intersects

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

abaabab \( \rightarrow \) \( (aa, 3) (ab, 1) (ab, 4) (ab, 6) (ba, 2) (ba, 5) \)  

min-prefix

xxyabab \( \rightarrow \) \( (xx, 1) (xy, 2) (ab, 4) \)  

min-prefix

- ⇒ *less candidates*

- *Count filtering* is a special case of *location-based mismatch filtering*
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* ≤ $2 \times d$

$q=5$, $d = 2$

\[
\begin{array}{ccccccccccc}
  c_1 & c_2 & c_3 & c_4 & c_5 & c_6 & c_7 & c_8 & c_9 & c_{10} & \ldots & c_{25} \\
  c_1 & c_2 & c_3 & c_4 & z_1 & z_2 & z_3 & z_4 & z_5 & c_{10} & \ldots & c_{25}
\end{array}
\]

*Probing Window*
Optimal q-gram Length

<table>
<thead>
<tr>
<th>q-grams</th>
<th>Time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-gram</td>
<td></td>
</tr>
<tr>
<td>3-gram</td>
<td></td>
</tr>
<tr>
<td>4-gram</td>
<td></td>
</tr>
<tr>
<td>6-gram</td>
<td></td>
</tr>
<tr>
<td>8-gram</td>
<td></td>
</tr>
<tr>
<td>10-gram</td>
<td></td>
</tr>
</tbody>
</table>

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 $O(d \times \Sigma^k \times \log(n\Sigma^k))$ similarity query time and $O(n\Sigma^k)$ space.

+ divide and conquer based on pigeon hole principle
  - Hamming Distance Join [Manku et al, WWW07]
  - PartEnum [Arasu et al, VLDB06]
Partitioning-based Approaches

- **Enumeration + Divide and conquer**
  - Hamming Distance Join \([\text{Manku et al, WWW07}]\)
  - PartEnum \([\text{Arasu et al, VLDB06}]\)
  - 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 \([1 .. f]\)

\[ f = 6, \ t = 2 \]

**Solution:** partition

\[ q = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 \end{bmatrix} \]

\[ v_1 = \begin{bmatrix} 1 & 1 & 0 & 1 & 1 & 1 \end{bmatrix} \]

- Duplicate data 3 times (or index)
- \(|C_i| = N / 2^2\)
- Elements in \( C_i \) need further verification

How many partitions are preserved?

\[ \binom{3}{1} \]

\[ \binom{2}{1} \]

### Problem:
- \#Duplication = \( C(n, \ t) \)
- \(|Cand_i| = N / 2^{(n-t)c}\)
- Cannot deal with large \( t \)
PartEnum [Arasu et al, VLDB06]

- **Part + Part + Enum**

  - form 2 partitions
  - \[ f = 6, \ t = 3 \]

  - v
    
    | 1 | 1 | 0 | 1 | 0 | 0 |

  - form 3 2\textsuperscript{nd}-level partitions

  - 1 1 0 → \{ 1?? , ?10, ??0 \}

  - 1 0 0 → \{ 1?? , ?0?, ??0 \}

  - At least one partition has error \( \lceil t/2 \rceil = 1 \)

  - Pigeon hole principle

- **Part** (n1=2 partitions)

  - Each record generate \( n1 \binom{n2}{k/n1} \) signatures

  - Hamming\((u, v) \leq k \Rightarrow \text{sigs}(u) \cap \text{sigs}(v) \neq \emptyset \)

- **Enum** (n2=3 partitions)

  - t = 10
  
    - ENUM with n=12 \( \Rightarrow 66 \) sigs / record
    - PartEnum with n1=3, n2=4 \( \Rightarrow 12 \) sigs / record
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
C. Similarity Join

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

2. Approximate algorithm
The Idea

- X and Y are very similar \(\Rightarrow\) \text{partOf}(X)\ is “very very similar” to \text{partOf}(Y)
Locality Sensitive Hashing

- LSH solves nearest neighbor problem approximately \[\text{Indyk & Motwani, STOC98} \] \[\text{Gionis et al, VLDB99} \] \[\text{Indyk FOCS00} \] ... \[\text{Andoni & Indyk, FOCS06} \]
  - 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 \( d(x, y) \leq R \) \( \Rightarrow \) \( \Pr[H[h(x) = h(y)]] \geq p_1 \)
  - if \( d(x, y) \geq cR \) \( \Rightarrow \) \( \Pr[H[h(x) = h(y)]] \leq p_2 \)

\( c > 1 \)
\( p_1 > p_2 \)
LSH Cookbook

- **Known LSH families**
  - $\mathcal{R}^d$, Hamming distance
    - $h_k(x) = x_k$, i.e., random projection on one dimension
  - $\mathcal{R}^d$, $L_1$ distance
  - $\mathcal{R}^d$, $L_p$ distance
    - $h_{r,b} = \lfloor (r \cdot x + b) / w \rfloor$, $r[i]$ 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 $e_i \in R$
    - select the first element after permutation
      $$\text{hash}(e_i)$$
      $$\text{sig}(R) = \min_i \{ \text{hash}(e_i) \}$$
Shingling Example

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

- **Doc D** → set of Shingles (aka. q-grams)
  - \( \text{Sim}(D_i, D_j) = \text{Jaccard}(\text{Shingles}(D_i), \text{Shingles}(D_j)) \)
- **Doc D** → set of signatures (of shingles)
  - \( \text{Sim}(D_i, D_j) = \frac{\text{Overlap}(\text{sig}s(D_i), \text{sig}s(D_j))}{N} \)

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
  
  \( h_u(x) = \text{sign}(u \cdot x) \), 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 → set of tokens with idf weighting → 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 $v(D)$ → $r(D)$
  - 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
Approximate Join Algorithm Without Quality Guarantee

- **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

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

- **Bigram Indexing** [Christen & Churches, Feb., 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. $[d_1, \text{SHA}_1], [d_2, \text{SHA}_2], \ldots$
   - collision on SHA digest values $\rightarrow$ near duplicate document

- 18K Web docs $\rightarrow$ 83 sec (I-Match) vs $\sim$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>
  - \( \text{Sig(till\_here)} = \{ \text{“are” \rightarrow “serve”, “to” \rightarrow “methods”, “to” \rightarrow “till\_here”} \} \)
  - \( \text{sim}(X \Rightarrow Y) = \frac{|\text{sig}(X) \cap \text{sig}(X)|}{|\text{sig}(X)|} \)
Roadmap

A. Motivation
B. Problem Definition and Scope
C. Similarity Join Algorithms
D. Epilogue
   1. Recurring ideas
   2. Performance comparison
   3. 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
Performance: Vector Space

- Corel: ColorHistogram & LayoutHistogram (68K points, 32d)
Performance: Metric Space

- Sentences, edit distance. Measures speedups
Performance: Set Similarity Join

- 800K DBLP, 14 tokens

![Graph showing time vs. cosine similarity for different methods: ProbeCount-sort, All-Pairs-Unsorted, All-Pairs. The graph illustrates the performance comparison with time given on a logarithmic scale.](image)
Performance: Set Similarity Join

- 800K DBLP, 14 tokens

Exact sim join algorithm is even faster than an approximate one!
Performance: Set Similarity Join

- 873K DBLP, 14 tokens

![Graph showing performance metrics](image)
Performance: Edit Similarity Join /1

- 366K UNIREF protein sequences, 465 chars, |Σ|=25
Performance: Edit Similarity Join \( /2 \)

- 863K DBLP, 105 chars, \( |\Sigma| = 93 \)

![Graph showing time in seconds for different Edit Distances and phases for \( \tau = 1, 2, 3 \).]
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

- Think out of the square
  - Black-box style similarity function
    - e.g., from the output of a classifier [Chandel et al, SIGMOD07]
    - e.g., IR relevance model that depends on many parameters
  - e.g., similarity function that depends on external parameters
- 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

Wei Wang: http://www.cse.unsw.edu.au/~weiw
Slides: http://www.cse.unsw.edu.au/~weiw/project/simjoin.html