

Similarity Search in Large Databases

Introduction to Similarity Search

Nikolaus Augsten

nikolaus.augsten@sbg.ac.at
Department of Computer Sciences
University of Salzburg



WS 2021/22

Version October 26, 2021

Outline

- 1 Similarity Search
 - Intuition
 - Applications
 - Framework

What is Similarity Search?

- Similarity search deals with the question:

How similar are two objects?

- “Objects” may be
 - strings (Augsten ↔ Augusten)
 - tuples in a relational database

(Augsten | Dominikanerplatz 3 | 204 | 70188)

↔

(N. Augusten | Dominikanerpl. 3 | @ | 70188)

- documents (e.g., HTML or XML)
- ...
- “Similar” is application dependant

Application I: Object Identification

- Problem:
 - Two data items represent the same real world object (e.g., the same person),
 - but they are represented differently in the database(s).
- How can this happen?
 - different coding conventions (e.g., Gilmstrasse, Hermann-von-Gilm-Str.)
 - spelling mistakes (e.g., Untervigil, Untervigli)
 - outdated values (e.g., Siegesplatz used to be Friedensplatz).
 - incomplete/incorrect values (e.g., missing or wrong apartment number in residential address).
- Focus in this course!

Application I: Flavors of Object Identification

- Duplicate Detection
 - one table
 - find all tuples in the table that represent the same thing in the real world
 - Example: Two companies merge and must build a single customer database.
- Similarity Join
 - two tables
 - join all tuples with similar values in the join attributes
 - Example: In order to detect tax fraud, data from different databases need to be linked.
- Similarity Lookup
 - one table, one tuple
 - find the tuple in the table that matches the given tuple best
 - Example: Do we already have customer X in the database?

Application II: Computational Biology

- DNA and protein sequences
 - modelled as text over alphabet (e.g. $\{A, C, G, T\}$ in DNA)
- Application: Search for a pattern in the text
 - look for given feature in DNA
 - compare two DNAs
 - decode DNA
- Problem: Exact matches fail
 - experimental measures have errors
 - small changes that are not relevant
 - mutations
- Solution: Similarity search
 - Search for *similar* patterns
 - *How similar* are the patterns that you found?

Application III: Error Correction in Signal Processing

- Application: Transmit text signal over physical channel
- Problem: Transmission may introduce errors
- Goal: Restore original (sent) message
- Solution: Find correct text that is closest to received message.

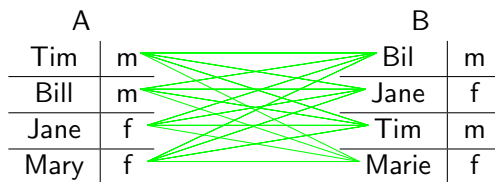
Framework for Similarity Search

1. Preprocessing (e.g., lowercase `Augsten` → `augsten`)
2. Search Space Reduction
 - Blocking
 - Sorted-Neighborhood
 - Filtering (Pruning)
3. Compute Distances
4. Find Matches

Search Space Reduction: Brute Force

- We consider the example of similarity join.
- Similarity Join:** Find all pairs of similar tuples in tables *A* and *B*.
 - Search space: $A \times B$ (all possible pairs of tuples)
 - Complexity: compute $|A||B|$ distances \rightarrow **expensive!**
 ($|A| = 30k$, $|B| = 40k$, $1ms$ per distance \Rightarrow join runs 2 weeks)

- Example:** 16 distance computations!

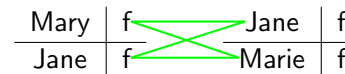
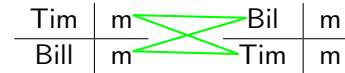


- Goal:** Reduce search space!

Search Space Reduction: Blocking

- Blocking**
 - Partition *A* and *B* into blocks (e.g., group by chosen attribute).
 - Compare only tuples within blocks.

- Example:** Block by gender (m/f):

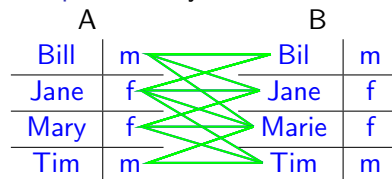


- Improvement:** 8 distance computations (instead of 16)!

Search Space Reduction: Sorted Neighborhood

- Sorted Neighborhood**
 - Sort *A* and *B* (e.g., by one of the attributes).
 - Move a window of fixed size over *A* and *B*.
 - move *A*-window if sort attribute of next tuple in *A* is smaller than in *B*
 - otherwise move *B*-window
 - Compare only tuples within the windows.

- Example:** Sort by name, use window of size 2:

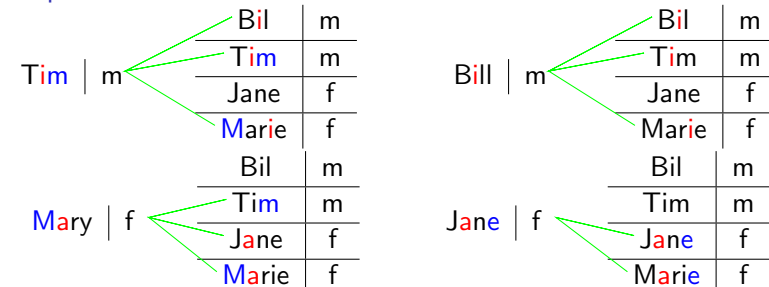


- Improvement:** 12 distance computations (instead of 16)!

Search Space Reduction: Filtering

- Filtering (Pruning)**
 - Remove (filter) tuples that cannot match, then compute the distances.
 - Idea: filter is faster than distance function.

- Example:** Do not match names that have no character in common:



- Improvement:** 11 distance computations (instead of 16)!

Distance Computation

Definition (Distance Function)

Given two sets of objects, A and B , a *distance function* for A and B maps each pair $(a, b) \in A \times B$ to a positive real number (including zero).

$$\delta : A \times B \rightarrow \mathbb{R}_0^+$$

- We will define distance functions for
 - sets
 - strings
 - ordered, labeled trees
 - unordered, labeled trees

Finding Matches: Threshold

	b_1	b_2	b_3
a_1	6	5	4
a_2	2	2	1
a_3	1	3	0

- Once we know the distances – which objects match?
- **Threshold Approach:**
 - fix threshold τ
 - algorithm:
 - foreach** $d_{ij} \in D$ **do**
 - if** $d_{ij} < \tau$ **then** match (a_i, b_j)
 - produces $n:m$ -matches
- **Example** with $\tau = 3$: $\{(a_2, b_1), (a_2, b_2), (a_2, b_3), (a_3, b_1), (a_3, b_3)\}$

Distance Matrix

Definition (Distance Matrix)

Given a distance function δ for two sets of objects, $A = \{a_1, \dots, a_n\}$ and $B = \{b_1, \dots, b_m\}$.

The *distance matrix* D is an $n \times m$ -matrix with

$$d_{ij} = \delta(a_i, b_j),$$

where d_{ij} is the element at the i -th row and the j -th column of D .

- Example distance matrix, $A = \{a_1, a_2, a_3\}$, $B = \{b_1, b_2, b_3\}$:

	b_1	b_2	b_3
a_1	6	5	4
a_2	2	2	1
a_3	1	3	0

Finding Matches: Global Greedy

- **Global Greedy Approach:**

- algorithm:
 - $M \leftarrow \emptyset$
 - $A \leftarrow \{a_1, a_2, \dots, a_n\}; B \leftarrow \{b_1, b_2, \dots, b_m\}$
 - create sorted list L with all $d_{ij} \in D$
 - while** $A \neq \emptyset$ **and** $B \neq \emptyset$ **do**
 - $d_{ij} \leftarrow$ deque smallest element from L
 - if** $a_i \in A$ **and** $b_j \in B$ **then**
 - $M \leftarrow M \cup (a_i, b_j)$
 - remove a_i from A and b_j from B
 - return** M

	b_1	b_2	b_3
a_1	6	5	4
a_2	2	2	1
a_3	1	3	0

- produces 1:1-matches
- must deal with tie distances when sorting $L!$ (e.g. sort randomly, sort by i and j)
- **Example** (sort ties by i, j): $\{(a_3, b_3), (a_2, b_1), (a_1, b_2)\}$

Overview: Finding Matches

	b_1	b_2	b_3
a_1	6	5	4
a_2	2	2	1
a_3	1	3	0

- **Threshold Approach:**
 - all objects with distance below τ match
 - produces $n:m$ -matches
 - threshold approach for our example with $\tau = 3$:
 $\{(a_2, b_1), (a_2, b_2), (a_2, b_3), (a_3, b_1), (a_3, b_3)\}$
- **Global Greedy Approach:**
 - pair with smallest distance is chosen first
 - produces 1:1-matches
 - global greedy approach for our example:
 $\{(a_3, b_3), (a_2, b_1), (a_1, b_2)\}$

Conclusion

- Framework for similarity queries:
 1. preprocessing
 - blocking
 - sorted-neighborhood
 - filtering (pruning)
 2. search space reduction
 3. compute distances: when are two objects similar?
 4. find matches: threshold, global greedy