

# Similarity Search in Large Databases

## Introduction to Similarity Search

Nikolaus Augsten

nikolaus.augsten@plus.ac.at  
Department of Computer Science  
University of Salzburg



<https://dbresearch.uni-salzburg.at>

WS 2024/25

Version October 1, 2024

# 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. Augsten | Dominikanerpl. 3 | @ | 70188)

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

# Application I: Entity Resolution (ER)

- Problem: (also known as “object identification”)
  - Two data items represent the same real world object/entity (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  - outdated values (e.g., `Siegesplatz` used to be `Friedensplatz`).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 Entity Resolution

- Duplicate Detection (also: “dirty ER”)
  - 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 (also: “clean-clean ER”)
  - 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?

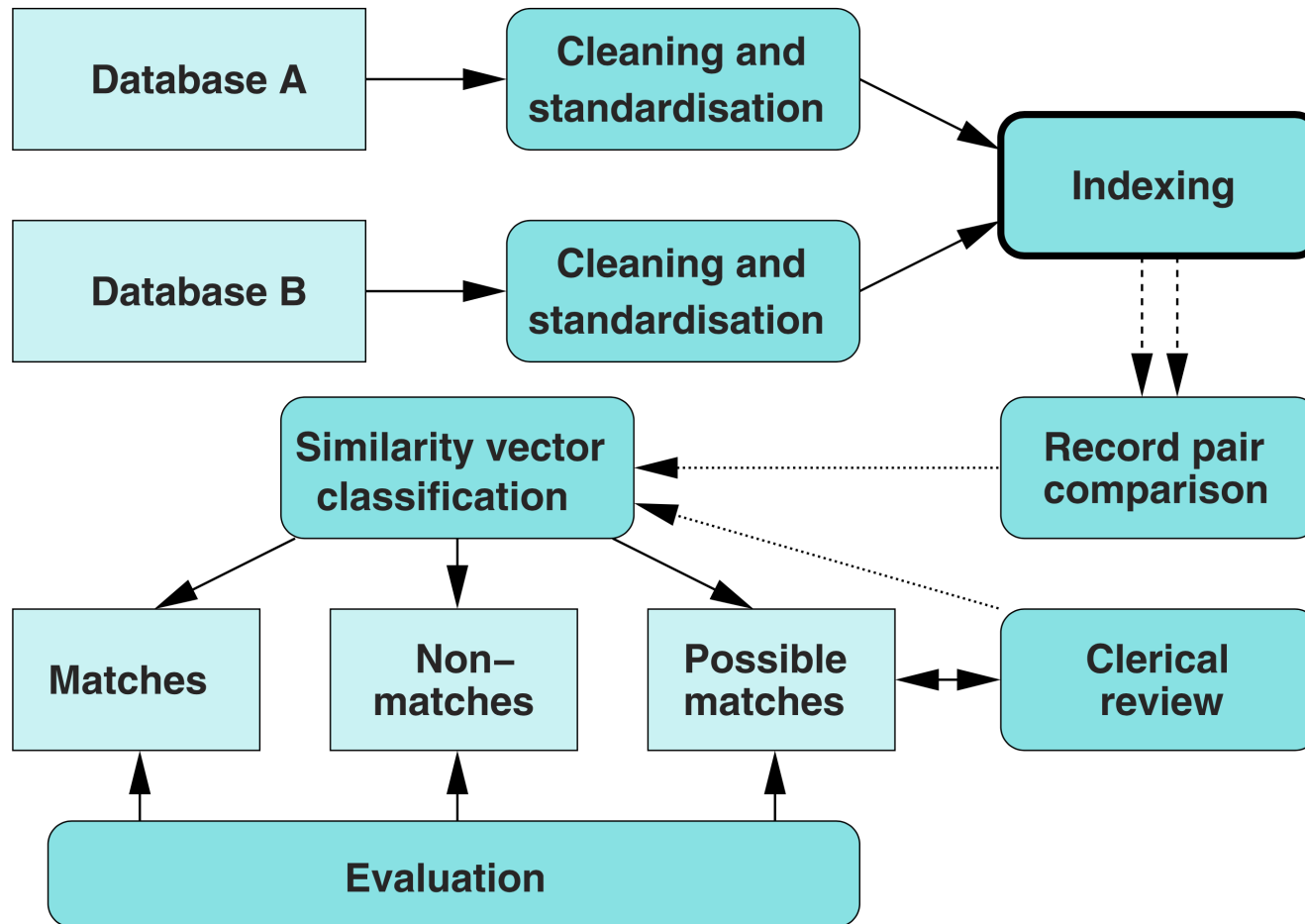
# Application II: Computational Biology

- 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



Source figure: [Chr12].

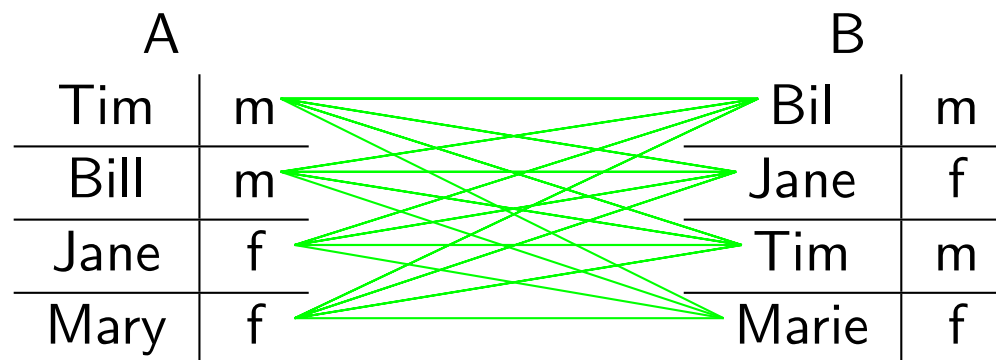


# Framework for Similarity Search / 2

1. Preprocessing: Cleaning and standardization, e.g.
  - lowercase all values: `Augsten` → `augsten`
  - standardize values: `{f,w,female,weiblich}` → `{f}`
  - standardize encoding: `K.Wolf Strasse` → `Karl-Wolf-Str.`
2. Indexing for Search Space Reduction
  - blocking
  - sorted-neighborhood
  - filtering (pruning)
  - nearest neighbor search
3. Compute Distances
  - compare record/tuple pairs
4. Find Matches: Classification
  - classify record/tuple pairs based on a distance or a vector of distances

# 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 will run for 2 weeks)
- **Example:** 16 distance computations!

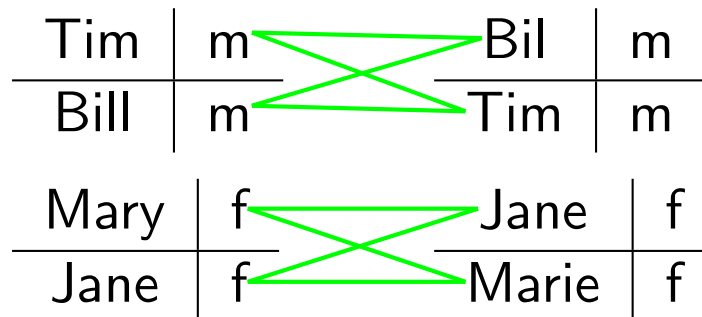


- **Goal:** Reduce search space!

# Search Space Reduction: Blocking

- **Blocking**
  - Generate blocks for  $A$  and  $B$  (seperately):
    1. compute one or more block keys for each tuple
    2. each distinct block key represents one block
    3. assign each tuple to all blocks of its block keys
  - Compare only tuples of block pairs with the same block key.
- **Example:** Attribute blocking: block key is value of a chosen attribute.

Block by “gender” attribute:



- **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:

| A    |   | B     |   |
|------|---|-------|---|
| Bill | m | Bil   | m |
| Jane | f | Jane  | f |
| Mary | f | Marie | f |
| Tim  | m | Tim   | m |

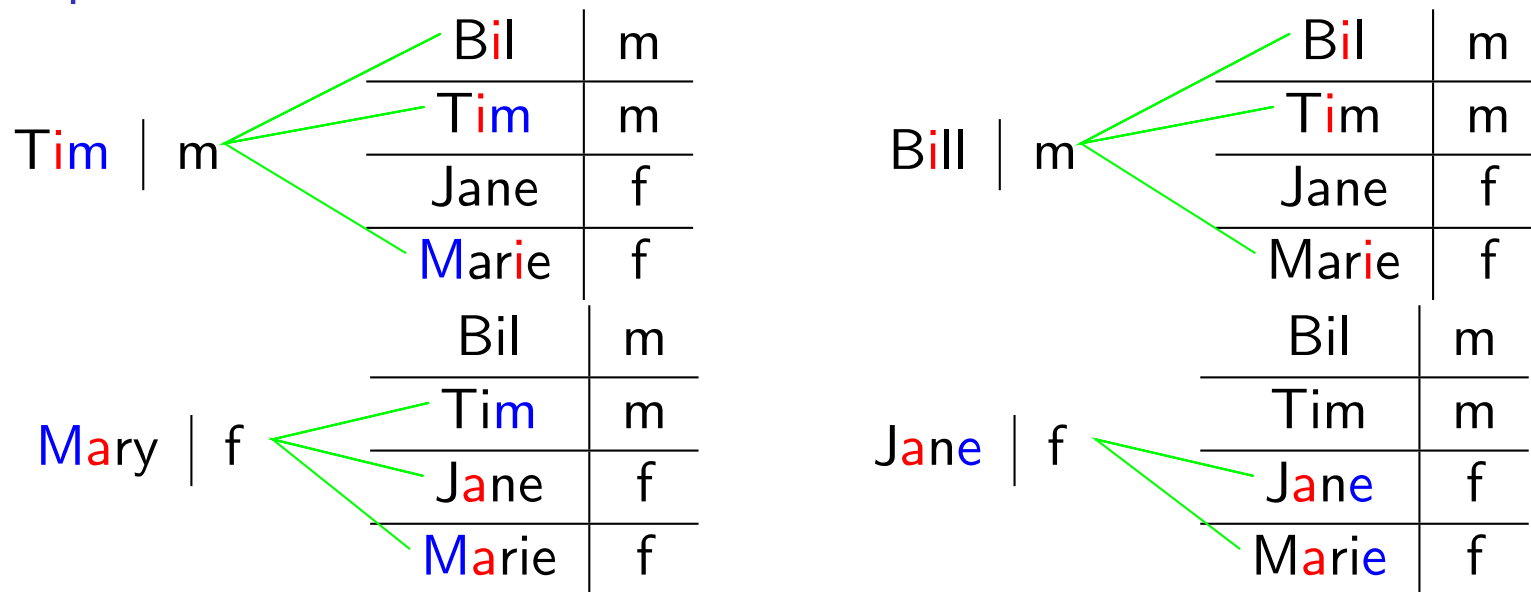
- Improvement: 12 distance computations (instead of 16)!

# Search Space Reduction: Filtering

- Filtering (Pruning)

- Filter: quick check if tuple pair is “promising” (i.e., distance is expected to be small enough).
- Do not compute distance function if filter check fails.
- Idea: filter is faster than distance function. Overhead for filtering is amortized by avoided distance computations.

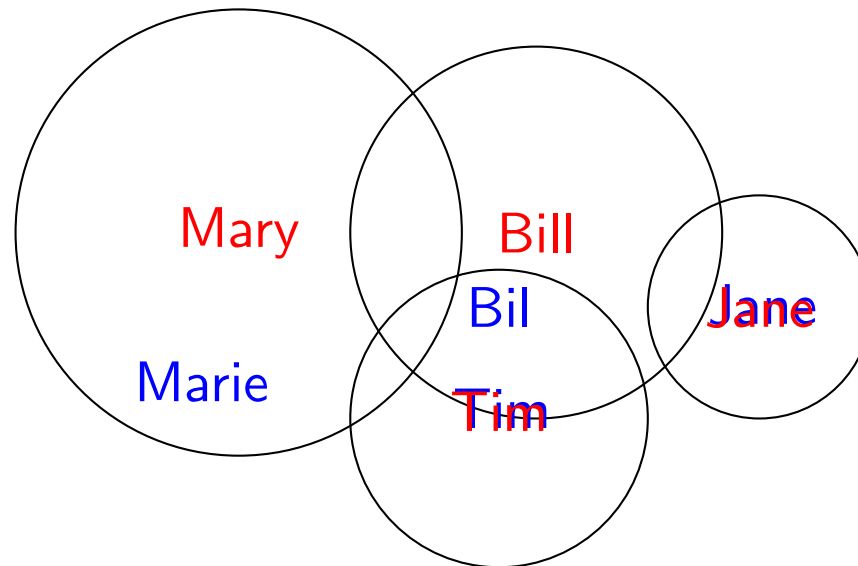
- Example: Do not match names that have no character in common:



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

# Search Space Reduction: Nearest-Neighbor Search

- Represent tuples in some vector space or metric space.
- For each tuple  $t$  in  $A$ :
  - search for all tuples in  $B$  that are close to tuple  $t$  of  $A$
  - distance to  $t$  is evaluated only for these tuples of  $B$
- Example: Search for **tuples of  $A$**  in **tuples of  $B$** .



- **Improvement:** 6 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
  - trees

# Distance Matrix

## Definition (Distance Matrix)

Given a distance function  $\delta$  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

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

- Once we know the distances – which objects match?
- Distance matrix and search space reduction:
  - matrix may be only partially filled at the time of computing matches
  - missing distance values are treated as infinite

# Finding Matches: Threshold Matching

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

- **Threshold Approach:**
  - fix threshold  $\tau$
  - for a given object, all objects with a distance range of  $\tau$  are matched
- **Algorithm:** produces  $n:m$  matching
  - foreach**  $d_{ij} \in D$  **do**
  - if**  $d_{ij} \leq \tau$  **then** match  $(a_i, b_j)$
- **Example** with  $\tau = 2$ :  $\{(a_2, b_1), (a_2, b_2), (a_2, b_3), (a_3, b_1), (a_3, b_3)\}$

# Finding Matches: k-Nearest Neighbor Matching

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

- **k-Nearest-Neighbor (kNN) Matching:**
  - fix number of neighbors  $k$
  - for a given object, form a match with its  $k$  nearest neighbors
- **Algorithm:** produces 1:k matching
  - foreach** row  $i$  of distance matrix  $D$  **do**
    - find column IDs  $c_1, c_2, \dots, c_k$  of  $k$  smallest values in row  $i$  of  $D$
    - form  $k$  matches  $(a_i, b_{c_1}), (a_i, b_{c_2}), \dots, (a_i, b_{c_k})$
- **Properties:**
  - not symmetric: transposed matrix  $D^T$  may give a different result
  - ties affect the matching (only for  $k$ -th neighbor)
- **Example**  $k = 2$ :  $\{(a_1, b_3), (a_1, b_2), (a_2, b_3), (a_2, b_1), (a_3, b_3), (a_3, b_1)\}$

# Finding Matches: Global Greedy

- **Global Greedy Approach**
  - form object pair with smallest distance first
  - matched objects are removed

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

- **Algorithm:** produces 1:1 matching

$M \leftarrow \emptyset$

$A \leftarrow \{a_1, a_2, \dots, a_n\}; B \leftarrow \{b_1, b_2, \dots, b_m\}$

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**  $M$

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

# Overview: Matching Techniques

- **Threshold Matching:**
  - all objects with distance within  $\tau$  match
  - $n:m$ -matching
  - symmetric, not affected by ties
- **$k$ -Nearest Neighbor Matching:**
  - each object is matched to its  $k$  closest objects
  - $1:k$ -matching
  - not symmetric, affected by ties
- **Global Greedy Approach:**
  - pair with smallest distance is matched first and removed
  - $1:1$ -matching
  - symmetric, affected by ties

# Conclusion

- Framework for similarity queries:
  1. Preprocessing: Cleaning and standardization
  2. Indexing for Search Space Reduction
    - blocking
    - sorted-neighborhood
    - filtering (pruning)
    - nearest neighbor search
  3. Compute Distances
  4. Find Matches: Classification
    - threshold-based
    - k-nearest-neighbor
    - global greedy



Peter Christen.

A survey of indexing techniques for scalable record linkage and deduplication.

*IEEE Trans. Knowl. Data Eng.*, 24(9):1537–1555, 2012.