A Problem at Our Municipality of Bozen

Given:
- reality owners DB (name and address of the reality)
- residents DB (name and residential address)
- both DBs cover the same geographic area (the city of Bozen)

<table>
<thead>
<tr>
<th>Owners (dataset A)</th>
<th>Residents (dataset B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peter Gilmstrasse 1</td>
<td>Rosa Siegesplatz 3/-/3</td>
</tr>
<tr>
<td>Arturas Gilmstrasse 3</td>
<td>Dario Friedhofplatz 4</td>
</tr>
<tr>
<td>Linas Marieng. 1/A</td>
<td>Romans Untervigil 1</td>
</tr>
<tr>
<td>Markus Cimitero 4</td>
<td>Adriano Mariengasse 1</td>
</tr>
<tr>
<td>Michael Gilmstrasse 5</td>
<td>Maria Siegesplatz 3/-/2</td>
</tr>
<tr>
<td>Igor Friedensplatz 2/A/1</td>
<td>Arturas Hermann-von-Gilm-Str. 3/A</td>
</tr>
<tr>
<td>Andrej Friedensplatz 3</td>
<td>Peter Hermann-von-Gilm-Str. 1</td>
</tr>
<tr>
<td>Francesco Untervigil 1</td>
<td>Markus Siegesplatz 2/A</td>
</tr>
<tr>
<td>Johann Cimitero 6/B</td>
<td>Juozas Hermann-von-Gilm-Str. 3/B</td>
</tr>
<tr>
<td>Igor Friedensplatz 2/A/2</td>
<td>Andrej Siegesplatz 3/-/1</td>
</tr>
<tr>
<td>Nikolaus Cimitero 6/A</td>
<td>Luigi Friedhofplatz 6</td>
</tr>
</tbody>
</table>

Query: Give me owner and resident for each apartment in Bozen!
What is Similarity Search?

- Similarity search deals with the question: 
  **How similar are two objects?**

- "Objects" can 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 dependent

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  - outdated values (e.g., Siegesplatz used to be Friedensplatz).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 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**

- **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 → expensive!
    \(|A| = 30k, |B| = 40k, 1ms\) per distance ⇒ join runs 2 weeks
  - **Example:** 16 distance computations!

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tim</td>
<td>m</td>
</tr>
<tr>
<td>Bill</td>
<td>m</td>
</tr>
<tr>
<td>Mary</td>
<td>f</td>
</tr>
<tr>
<td>Jane</td>
<td>m</td>
</tr>
</tbody>
</table>

- **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):

<table>
<thead>
<tr>
<th>Tim</th>
<th>m</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bill</td>
<td>m</td>
</tr>
<tr>
<td>Mary</td>
<td>f</td>
</tr>
<tr>
<td>Jane</td>
<td>f</td>
</tr>
<tr>
<td>Marie</td>
<td>f</td>
</tr>
</tbody>
</table>

- **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  
  Jane | f  
  Mary | f  
  Tim  | m  

  $B$ | m  
  ---|---
  Bill | m  
  Jane | f  
  Marie | f  
  Tim  | m  

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

Search Space Reduction: Filtering

- **Filtering (Pruning)**
  - Remove (filter) tuples that can not match, then compute the distances.
    - Idea: filter is faster than distance function.
- **Example:** Do not match names that have no character in common:
  
  $A$ | $m$
  ---|---
  Tim | m  
  Jane | f  
  Marie | f  
  Tim  | m  

  $B$ | m  
  ---|---
  Bill | m  
  Jane | f  
  Marie | f  
  Jane  | f  

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

Distance Computation

- **Definition (Distance Function)**
  - Given two sets of objects, $A$ and $B$, a distance function $\delta$ for $A$ and $B$ maps each pair $(a, b) \in A \times B$ to a positive real number (including zero).
    
    \[ \delta : A \times B \rightarrow \mathbb{R}^+ \]
  - We will define distance functions for:
    - strings
    - ordered, labeled trees
    - unordered, labeled trees

Distance Matrix

- **Definition (Distance Matrix)**
  - Given a distance function $\delta$ for two sets of objects, $A = \{a_1, \ldots, a_n\}$ and $B = \{b_1, \ldots, b_m\}$.
  - 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, $A = \{a_1, a_2, a_3\}$, $B = \{b_1, b_2, b_3\}$:**
    
    \[
    \begin{array}{ccc}
    b_1 & b_2 & b_3 \\
    a_1 & 6 & 5 & 4 \\
    a_2 & 2 & 2 & 1 \\
    a_3 & 1 & 3 & 0 \\
    \end{array}
    \]
Once we know the distances – which objects match?

Threshold Approach:
- fix threshold \( \tau \)
- algorithm:
  ```
  \textit{foreach} d_{ij} \in D \text{ do}
  \quad \text{if} \ d_{ij} < \tau \ \text{then} \ \text{match} \ (a_i, b_j)
  ```
- produces \( n:m \)-matches

Example with \( \tau = 3 \):
\[
\{(a_2, b_1), (a_2, b_2), (a_2, b_3), (a_3, b_1), (a_3, b_3)\}
\]

Global Greedy Approach:
- pair with smallest distance is chosen first
- produces 1:1-matches
- global greedy approach for our example:
\[
\{(a_3, b_3), (a_2, b_1), (a_1, b_2)\}
\]

Overview: Finding Matches

<table>
<thead>
<tr>
<th></th>
<th>( b_1 )</th>
<th>( b_2 )</th>
<th>( b_3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a_1 )</td>
<td>6</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>( a_2 )</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>( a_3 )</td>
<td>1</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>

Threshold Approach:
- all objects with distance below \( \tau \) match
- produces \( n:m \)-matches
- threshold approach for our example with \( \tau = 3 \):
\[
\{(a_2, b_1), (a_2, b_2), (a_2, b_3), (a_3, b_1), (a_3, b_3)\}
\]

Outline

1. Similarity Search
   - Intuition
   - Applications
   - Framework

2. Organization of the Course
   - Course Structure
   - Project
   - Course Overview

3. Demo: Similarity Join on Residential Addresses
   - Problem Definition
   - Street Matching
Organization of the Course

Course Structure

About this Course

- Lecture: Thursday, 10:30-12:30 (room E411)
- Lab: Thursday, 9:30-10:30 (room E431) – starts next week
- Office hours: Friday, 8:45-10:00 (office 2.19)
- Course info: [http://www.inf.unibz.it/dis/teaching/SS](http://www.inf.unibz.it/dis/teaching/SS)
- Feedback most welcome! (augsten@inf.unibz.it or orally)

How to Complete the Course Successfully?

Part I: Project (groups of 2–3 people)
- hand in Java source code
- hand in project report
- give a presentation of the project

Part II: Oral exam (15 minutes)

Final Grade
- pass both parts
- each part is graded with up to 30 points
- final grade = grade project × 0.4 + grade oral exam × 0.6

Project Report

- Important deliverable of the project!
- Elements:
  - Motivation
  - Problem definition
  - Short introduction to the solution (1-2 paragraphs per method)
  - Experiments and discussion
  - Manual (installation and usage instruction for your code deliverable)
- Approximately 6 pages
- Be concise!

Project Milestones

- **Milestone 1** — March 22 (Lab 3)
  - form groups
  - choose project
- **Milestone 2** — May 3 (Lab 8)
  - demo your implementation
  - discuss report (without experimental part)
- **Milestone 3** — June 7 (Lab 12)
  - hand in final version of report
  - presentation
About the Lab

- You work on your project.
- As the need arises:
  - discussion of project related problems
  - discussion of course contents
- First lab (March 8)
  - project proposals will be introduced
  - working environment will be set up
- Last lab (June 7)
  - project presentation

Assumptions for the Solutions in this Course

- Large data volumes
  - can not be done by hand
  - solution must be efficient
- Data-driven, not Process-driven
  - Sometimes it is better to change the world, e.g., force people to adhere to coding conventions, instead of fixing the errors later.
  - We can not change the world.
- No domain-specific solution (e.g., address standardization)
- No training phase (e.g., supervised learning)
- No expensive configuration (e.g., define dictionaries, rules)
- Tuning parameters (like weights) are OK

Contents

- Introduction: Similarity Search
- Flat Data (Strings):
  - String Edit Distance: Algorithm and Complexity
  - q-Gram Distance: Filter for the String Edit Distance
- Hierarchical Data (Trees):
  - Hierarchical Data and XML in Relational Database Systems
  - Tree Edit Distance: Algorithms and Complexity
  - Filters for the Tree Edit Distance
  - Pruning with Reference Sets
  - Tree Decompositions: Binary Branches and pq-Grams
  - Top-k Subtree Queries
  - Similarity of Unordered Trees (Data-Centric XML)

Outline

- Similarity Search
  - Intuition
  - Applications
  - Framework
- Organization of the Course
  - Course Structure
  - Project
  - Course Overview
- Demo: Similarity Join on Residential Addresses
  - Problem Definition
  - Street Matching
Back to Our Initial Example

- Given:
  - reality owners DB (name and address of the reality)
  - residents DB (name and residential address)
  - both DBs cover the same geographic area (the city of Bozen/Italy)

- Problem Definition

  - Fiedensplatz
  - Friedhofplatz
  - Untervigli

  - Similarity Search
    - String Similarity
      - Observation 1: Some street names are similar.
        - dataset A
          - Gilmstrasse
          - Friedensplatz
          - Cimitero
          - Untervigli
          - Marieng.
        - dataset B
          - Untervigli
          - Mariengasse
          - Gilmstrasse
          - Hermann-von-Gilm-Str.

        - We match:
          - Untervigli ↔ Untervigli
          - Marieng. ↔ Mariengasse
          - Gilmstrasse ↔ Hermann-von-Gilm-Str.

        - But what to do with the others?
          - Friedensplatz was renamed to Siegesplatz, but one database was not updated
          - Cimitero is the Italian name for Friedhofplatz (German name)

- Problem: Friedensplatz looks more like Friedhofplatz than like Siegesplatz!

Database Representation

<table>
<thead>
<tr>
<th>Owners (dataset A)</th>
<th>Residents (dataset B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peter Gilmstrasse 1</td>
<td>Anita Hermann-von-Gilm-Str. 6</td>
</tr>
<tr>
<td>Arturas Gilmstrasse 3</td>
<td>Linas Mariengasse 1/A</td>
</tr>
<tr>
<td>Markus Cimitero 4</td>
<td>Adriano Mariengasse 1</td>
</tr>
<tr>
<td>Michael Cimitero 5</td>
<td>Maria Siegesplatz 3/-/2</td>
</tr>
<tr>
<td>Igor Friedensplatz 2/A/1</td>
<td>Arturas Hermann-von-Gilm-Str. 3/A</td>
</tr>
<tr>
<td>Andrej Friedensplatz 3</td>
<td>Peter Hermann-von-Gilm-Str. 1</td>
</tr>
<tr>
<td>Francesco Untervigli 1</td>
<td>Markus Siegesplatz 2/A</td>
</tr>
<tr>
<td>Johann Cimitero 6/B</td>
<td>Lucjan Hermann-von-Gilm-Str. 3/B</td>
</tr>
<tr>
<td>Igor Friedensplatz 2/A/2</td>
<td>Andrej Siegesplatz 3/-/1</td>
</tr>
<tr>
<td>Nikolaus Cimitero 6/A</td>
<td>Luigi Friedensplatz 6</td>
</tr>
</tbody>
</table>

- Demonstration: String Similarity
  - Street name tables:
    - | strID | name   | num | entr | apt |
      - | α1   | Gilmstrasse | 1   |      |     |
      - | α2   | Friedensplatz | 2 A  | 1    |     |
      - | α3   | Gilmstrasse | 3   |      |     |
      - | α4   | Cimitero   | 4   |      |     |
      - | α5   | Untervigli | 1   |      |     |
      - | α6   | Marieng.   | 1   |      |     |
  - | strID | name   | num | entr | apt |
      - | β1   | Friedhofplatz  |   |     |     |
      - | β2   | Hermann-von-Gilm-Str. 3 | A |
      - | β3   | Siegesplatz  | 2 A |
      - | β4   | Mariengasse | 1 |
      - | β5   | Untervigli | 1 |

- Distance matrix for the q-gram distance between strings:

  - Matches with the global greedy algorithm:
    - \{(α2, β2), (α4, β4), (α5, β4), (α1, β2), (α3, β3)\}
**Observation 2:** Different streets have different addresses.

**Build address tree:**

<table>
<thead>
<tr>
<th>Street Name</th>
<th>strID</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gilmstrasse 1</td>
<td>α1</td>
<td>Gilmstrasse</td>
</tr>
<tr>
<td>Gilmstrasse 3</td>
<td>α2</td>
<td>Friedensplatz</td>
</tr>
<tr>
<td>Gilmstrasse 5</td>
<td>α3</td>
<td>Cimitero</td>
</tr>
<tr>
<td>Friedensplatz 2/A</td>
<td>α4</td>
<td>Untervigil</td>
</tr>
<tr>
<td>Friedensplatz 3</td>
<td>α5</td>
<td>Marieng.</td>
</tr>
<tr>
<td>Cimitero 4</td>
<td>β1</td>
<td>Herman-von-Gilm-Str.</td>
</tr>
<tr>
<td>Cimitero 6/A</td>
<td>β2</td>
<td>Friedhofplatz</td>
</tr>
<tr>
<td>Cimitero 6/B</td>
<td>β3</td>
<td>Untervigil</td>
</tr>
<tr>
<td>Untervigil 1</td>
<td>β4</td>
<td>Marieng.</td>
</tr>
<tr>
<td>Marieng. 1/A</td>
<td>β5</td>
<td></td>
</tr>
</tbody>
</table>

Address is path from root to leaf.

Example: Shaded path is the address *Friedensplatz 2/A/1* (house number 2, entrance A, apartment 1).

**Street name tables:**

<table>
<thead>
<tr>
<th>strID</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>α1</td>
<td>Gilmstrasse</td>
</tr>
<tr>
<td>α2</td>
<td>Friedensplatz</td>
</tr>
<tr>
<td>α3</td>
<td>Cimitero</td>
</tr>
<tr>
<td>α4</td>
<td>Untervigil</td>
</tr>
<tr>
<td>α5</td>
<td>Marieng.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>strID</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>β1</td>
<td>Herman-von-Gilm-Str.</td>
</tr>
<tr>
<td>β2</td>
<td>Friedhofplatz</td>
</tr>
<tr>
<td>β3</td>
<td>Untervigil</td>
</tr>
<tr>
<td>β4</td>
<td>Marieng.</td>
</tr>
<tr>
<td>β5</td>
<td></td>
</tr>
</tbody>
</table>

**Distance matrix for the pq-gram distance between trees:**

<table>
<thead>
<tr>
<th></th>
<th>β1</th>
<th>β2</th>
<th>β3</th>
<th>β4</th>
<th>β5</th>
</tr>
</thead>
<tbody>
<tr>
<td>α1</td>
<td>1.0</td>
<td>0.143</td>
<td>1.0</td>
<td>0.6667</td>
<td>0.6667</td>
</tr>
<tr>
<td>α2</td>
<td>1.0</td>
<td>1.0</td>
<td>0.5758</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>α3</td>
<td>0.4118</td>
<td>0.9167</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>α4</td>
<td>1.0</td>
<td>0.7647</td>
<td>1.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>α5</td>
<td>1.0</td>
<td>0.9</td>
<td>1.0</td>
<td>0.4545</td>
<td>0.4545</td>
</tr>
</tbody>
</table>

Matches with the global greedy algorithm:

{((α4, β4), (α3, β3), (α5, β5), (α2, β2), (α1, β1))}

**Combining String and Tree Distance**

- Use strings and trees!
- String distance *s*, tree distance *t*
- Weight *ω* ∈ [0..1]
  - *ω* = 0 → only trees
  - *ω* = 1 → only strings
- Overall distance *d* (using weighted Euclidean distance):
  \[ d = \sqrt{\omega s^2 + (1 - \omega)t^2} \]
Experiments: Results for Real World Data

- Similarity join on three real databases:
  - electricity company (elec) – German street names, 45k addresses
  - registration office (reg) – Italian street names, 43k addresses
  - census database (cens) – German street names, 11k addresses

- Measure precision and recall
  - Precision: correctly computed matches to total number of computed matches
  - Recall: correctly computed matches to total number of correct matches

\[
\begin{array}{|c|c|c|c|c|c|}
\hline
\text{idA} & \text{idB} & \beta_1 & \beta_2 & \beta_3 & \beta_4 & \beta_5 \\
\hline
\alpha_1 & 1.0 & 0.7761 & 1.0 & 0.6796 & 0.8498 \\
\alpha_2 & 0.7654 & 1.0 & <0.5736 & 0.9649 & 1.0 \\
\alpha_3 & 0.7654 & 0.9782 & 1.0 & 0.9392 & 1.0 \\
\alpha_4 & 1.0 & 0.8584 & 1.0 & 0.7071 & <0.2357 \\
\alpha_5 & 0.9608 & 0.9199 & 1.0 & <0.4241 & 0.7767 \\
\hline
\end{array}
\]

\[
\text{map A | B}
\]

- Gilmstrasse ↔ Hermann-von-Gilm-Str.
- Friedensplatz ↔ Siegesplatz
- Cimitero ↔ Friedhofplatz
- Untervigil ↔ Untervigli
- Marieng. ↔ Mariengasse

Nikolaus Augsten (DIS) Similarity Search Unit 1 – March 1, 2012 37 / 40

Experiments: Results for Real World Data

- Similarity join with global greedy matching
- String weight \( \omega \) varies from 0 (only trees) to 1 (only strings)
- Measure precision and recall (high is good)

- Precision: correctly computed matches to total number of computed matches
- Recall: correctly computed matches to total number of correct matches

\[
\begin{array}{|c|c|}
\hline
\text{string weight} & \text{recall / precision} \\
\hline
0 & 1 \\
0.2 & 0.9 \\
0.4 & 0.8 \\
0.6 & 0.7 \\
0.8 & 0.6 \\
1 & 0.5 \\
\hline
\end{array}
\]

elec (German) ↔ reg (Italian)

Nikolaus Augsten (DIS) Similarity Search Unit 1 – March 1, 2012 38 / 40

Experiments: Results for Real World Data

- Similarity join with global greedy matching
- String weight \( \omega \) varies from 0 (only trees) to 1 (only strings)
- Measure precision and recall (high is good)

\[
\begin{array}{|c|c|}
\hline
\text{string weight} & \text{recall / precision} \\
\hline
0 & 1 \\
0.2 & 0.9 \\
0.4 & 0.8 \\
0.6 & 0.7 \\
0.8 & 0.6 \\
1 & 0.5 \\
\hline
\end{array}
\]

reg (Italian) ↔ cens (German)
Experiments: Results for Real World Data

- Similarity join with global greedy matching
- String weight $\omega$ varies from 0 (only trees) to 1 (only strings)
- Measure precision and recall (high is good)

Summary of the experimental results:
- High string weight $\omega$ good for German-German, bad for German-Italian
- String weight $\omega = 0.5$ good for both German-German and German-Italian
- Precision and recall very high ($\omega = 0.5$):
  - more than 90% even for German-Italian
  - precision almost 100%, recall 95% for German-German ($\omega = 0.5$)