Motivation

- Dataset: collection of photos
- Query input: a photo
- Task: find relevant photos w.r.t. query
Motivation

- Dataset: fingerprints
- Query input: fingerprint obtained by a fingerprint scanner
- Task: identify person

Multimedia Database

- Data: objects from a multimedia data type
  - Images
  - Audio files
  - Videos
  - 3D objects
  - Text
- Operations
  - Insertions, deletions
  - Search for data
Motivation

- Similarity search: find similar objects to the query input

Organization of the Tutorial

- Part I: The similarity problem
- Part II: Indexing
- Part III: Similarity Join
PART I
THE SIMILARITY PROBLEM

The similarity problem

- Concept of similarity is inherently subjective
  - Matching parts between objects
  - Optimization problem: how much work do I have to do to transform one object into another one?
- Definition of “Similarity Model”
  - Domain of objects
  - Similarity (or dissimilarity) function
The similarity problem

- **Metric space approach** [ZAD+06]
  - Universe of objects: $U$
  - Dissimilarity function: $\delta: U \times U \rightarrow \mathbb{R}$
    - Non-negativity: $\delta(x, y) \geq 0$
    - Reflexivity: $\delta(x, y) = 0 \iff x = y$
    - Symmetry: $\delta(x, y) = d(y, x)$
    - Triangular inequality: $\delta(x, z) \leq \delta(x, y) + \delta(y, z)$

---

The similarity problem

- **Example: vector spaces**
  - Domain $U = \mathbb{R}^d$
    - $d$ is the dimension of the space
  - Distance $\delta_{L_p}$ from Minkowski family of distances

$$\delta_{L_p}(x, y) = \left( \sum_{i=1}^{d} |x_i - y_i|^p \right)^{1/p}, p \geq 1 \quad p = 1 \text{ Manhattan distance}$$

$$p = 2 \text{ Euclidean distance}$$
The similarity problem

- Other common metric distance functions

  **Quadratic forms:**
  \[ \delta_{QF}(x, y) = \sqrt{(x - y)'A(x - y)} \]

  **Earth Mover’s distance:**
  \[ \delta_{EMD}(x, y) = \min \left\{ \sum_{i=1}^{d} \sum_{j=1}^{d} c_{ij}f_{ij} \right\} \]

  subject to
  \[ f_{ij} \geq 0 \]
  \[ \sum_{i=1}^{d} f_{ij} = y_j, \forall j = 1, \ldots, d \]
  \[ \sum_{j=1}^{d} f_{ij} = x_i, \forall i = 1, \ldots, d \]

Similarity search

- Search for “similar objects”
- Content-based search: query-by-example

  - **Range query** (give me the very similar ones – over 80%)
  - **k nearest neighbors query** (give me the 3 most similar)
Similarity search

- Range query

\[(q, r) = \{u_2, u_4, u_6, u_7\}\]

Similarity search

- k-NN query

\[3 - NN(q) = \{u_2, u_6, u_7\}\]
Similarity search

- Query-by-sketch

Figure from: Jose Saavedra and Benjamin Bustos. An improved histogram of edge local orientations for sketch-based image retrieval. In Proc. 32nd Annual Symposium of the German Association for Pattern Recognition (DAGM'10), LNCS 6376, pages 432-441, 2010.

Similarity search

- Efficiency
  - How much cost a similarity query
  - Can be measured as
    - CPU time
      - Number of distance computations
    - I/O time
      - Number of disk page accesses
Similarity search

- Effectiveness
  - Quality of the answer returned by a similarity search
  - Does the similarity model allow the user to retrieve relevant objects?

Effectiveness evaluation

- Measure the ability of the system to retrieve relevant objects, while discarding non-relevant objects
- Two aspects:
  - Ground truth
  - Evaluation metrics
Similarity search

- **Evaluation metrics**
  - Confusion matrix
    - True positives, True negatives, False positives, False negatives
  - Precision vs Recall curves
  - Mean average precision (MAP)
  - F-score
Similarity models

- Similarity models for images
  - Descriptors
    - Color histograms
    - Edge histograms
    - Morphological model
  - Problems
    - Image size
    - Noise
    - Occlusion

- Similarity models for 3D data
  - Descriptors
    - Global features (volumetric, surface, images)
    - Local features
  - Problems
    - Pose normalization
    - Noise
    - Holes
    - Non-rigid transformations
Similarity models

- Deep features (Example by Labrada)

Figure from: Arnul Labrada, Benjamin Bustos, and Ivan Sipiran. A convolutional architecture for 3D model embedding using image views. To appear in The Visual Computer.

PART II
INDEXING
Indexing

- How to compute a similarity query?
  - Given a dataset of size $n$ and a query object $q$
    - Compute all distances between $q$ and objects in dataset
      - So-called “sequential scan”
      - Complexity: $\Theta(n)$ distance computations
    - Problem: distance computation can be expensive
      - For vector spaces with dimensionality $d$
        - Minkowski distance: $\Theta(d)$
        - Quadratic forms: $\Theta(d^2)$
        - Earth Mover’s Distance: $O(d^2 \log d)$

Indexing

- Filter-and-refine
  - Let $\delta'$ be a distance that is “cheap” to compute and approximates $\delta$
    - Filter irrelevant objects using $\delta'$
    - Refine the candidate list using $\delta$
  - If $\delta'$ is a lower bound of $\delta$
    - Guaranteed that there will be no false negatives
Indexing: vector spaces

- Multidimensional index
  - Balanced trees (in general)
  - Each node corresponds to
    - A disk page / memory region
    - A space region
  - Two types of pages (nodes)
    - Data pages: leaves, contain data points
    - Directory pages: internal nodes that
      - Contain references to child nodes
      - Describe the spatial region of the child nodes

Indexing: vector spaces

- Basic structure
Indexing: vector spaces

- Spatial regions
  - Idea: store close points in the same data page or subtree
  - Can have different shapes
    - Bounding box
    - Hypersphere
    - Hypercube
    - Multidimensional cylinder
    - A combination of the previous shapes

- Hierarchy: a space region that defines a node must be completely inside the space region that defines the parent node
- Multidimensional indexes are dynamic structures (insertions, deletions)
Indexing: vector spaces

- Typical insertion procedure
  - Search for adequate data page (search new object, insert where it should have been found)
  - Add object to data page
  - In case of overflow, split node
  - Adjust space region information at parent node
  - If parent node overflows, split and proceed recursively
  - If root node split, create new root

---

Example: R-tree [Gut84]

Parameters:

- \( m = 2 \)
- \( M = 5 \)

![R-tree structure](image)

---

Indexing: vector spaces

- “Good” node split strategy?

or
Indexing: vector spaces

- "Good" node split strategy?

```
Bad strategy
or
Better strategy
```

Indexing: range queries

- Range queries with multidimensional index
Indexing: k-NN queries

- k-NN queries with multidimensional index
  - Result may be ambiguous
  - There is no radius value $r$ that allows us to discard space regions a priori
    - Depth search (1-NN case)
      - Start with $\text{distNN} = \infty$, use $\text{distNN}$ as search radius
      - Each time one computes a distance, update $\text{distNN}$ if a better candidate is found
      - Use $\text{distNN}$ to discard space regions
    - Problem: no guarantee that we visit the data pages in optimal order

- What if we knew a priori the value $r_{kNN}$ = distance to the k-th NN to the query?
  - We could use directly the range search algorithm
    - This would minimize the number of visited data nodes
  - But, in practice we do not know that value…
    - Maybe we could try to estimate it?
Indexing: k-NN queries

- k-NN queries with multidimensional index
  - Priority search algorithm by Hjaltason and Samet [HS95]
    - Search nodes in the order given by the lower bound distances of the space regions to the query object
    - Guarantee that only visit data nodes that intersect to query region with \( r = r_{kNN} \), without knowing \( r_{kNN} \) a priori
  - Range-optimal algorithm
  - There is no inherent advantage on knowing or estimating \( r_{kNN} \)

Indexing: cost model

- What is the cost of a similarity search using a multidimensional index?
- Cost model [Böh00]
  - Range queries
  - Square regions, no overlaps
  - Space: unitary hypercube $[0,1]^d$
    - Volume of space = 1
  - Points (data and queries) uniformly distributed


Indexing: cost model

- Probability of accessing a page if $r = 0$ (point query)
  - Volume of space region
- Probability of accessing a page if $r > 0$
  - Transform the range query to a point query
    - Trick: enlarge each space region by $r$
    - Minkowski sum
  - Probability depends exponentially on $d$
Indexing: cost model

- Probability of accessing a page if $r > 0$
  - Moreover, high-dimensional spaces are strange
    - 2-D intuition is misleading
    - Almost all the volume is located at the border
    - One needs higher values of $r$ to get some objects inside the query ball
      - Indeed, $r$ could be larger than 1
      - That means, part of query ball is outside the space
    - Meaning of NN in high-dimensional spaces?

Indexing: metric spaces

- Metric index: try to minimize number of distance computations
- Basic ideas
  - Partition the space into “regions”
  - Discard “regions” during the search
  - Search on non-discarded “regions”
- Two main families
  - Pivot-based indexing
  - Compact partitions
Indexing: pivot-based indexing

- Pivot-based indexing

Discarding criterion:

\[
|\delta(p_i, q) - \delta(p_i, u)| > r
\]

Indexing: pivot-based indexing

- Search cost
  - Distances from query to pivots, plus
  - Distances from query to non-discarded objects
- There is a trade-off between number of pivots and search cost
- How to select the pivots?
  - Great topic! Ask me please at the end of the tutorial
Indexing: pivot-based indexing

- k-NN search using pivot-based indexing
  - Use closest pivot as starting candidate, then use diminishing radius technique
  - There is a range-optimal algorithm for pivot-based indexing

Indexing: compact partitions

- Divide the space in “compact regions” (objects close to each other)
- Each region has a representative object $c$
- Each region can be recursively partitioned
- Regions can be defined by
  - Voronoi partition
  - Covering radius
Indexing: compact partitions

- Example: covering radius

![Diagram showing covering radius with circles and points]

M-tree [CPZ97]
- Balanced tree
- Dynamic index
- Secondary memory
- Discard branches by using lower-bound distances to regions

Indexing: compact partitions

- **M-tree structure**
  - **External nodes**
    - Feature values of data objects
    - Distance from data object to its parent “routing object”
  - **Internal nodes**
    - Reference to parent node
    - Routing objects \( O_r \), for each of them:
      - Feature value of \( O_r \)
      - Reference to subtree corresponding to \( O_r \)
      - Covering radius of \( O_r \)
      - Distance from \( O_r \) to its parent routing object

Indexing: summary

- Are index techniques efficient in practice?
  - Short answer: No
  - They only work for very small dimension \( d \)
  - For high enough \( d \): cost equal to sequential scan
- Is the only solution a linear scan?
  - My opinion: for exact search, maybe 😊
  - For the moment, one can resort to
    - Distributed algorithms
    - Approximate algorithms
Similarity join

- Given two datasets $D_1$ and $D_2$
  - Find all pairs $(x, y)$ such that $x \in D_1$, $y \in D_2$, and $x$ is similar to $y$ using similarity criteria $s$
- Similarity criteria can be
  - range-based, or
  - nearest neighbor-based
    - k-NN
**Similarity join**

- Computing a similarity join
  - Nested loop
  - Index the data first
    - An index for each similarity model
    - Static dataset vs dynamic dataset

**Self-similarity join**

- Particular case
  - $D = D_1 = D_2$
  - Let $n = |D|$ be the size of the dataset

- Naive algorithm for self-similarity join
  - Compute all pairs of distances (nested loop)
  - This costs $\theta(n^2)$ distance computations

- In practice, this is computationally expensive

*Can we improve the efficiency?*
Approximate self-similarity join

- Approximate technique [FBR20]
  - No guarantee of obtaining the exact result
  - Trade-off between efficiency and effectiveness
- Let $q \in D$ and $k \geq 1$ (integer)
  - $Q_k(D, q)$: exact answer of $k$-NN query for $q$
  - $Q_a(D, q, k)$: approximate result for $Q_k(D, q)$

$$\text{Precision} = \frac{|Q_k(D, q) \cap Q_a(D, q, k)|}{k}$$


Root-join algorithm: first idea

- Algorithm for 1-NN self-similarity join
Root-join algorithm: first idea

- Build $\sqrt{n}$ groups of size $\sqrt{n}$

For each object in the set, compute the NN only within its group

AMW 2023 - Tutorial: Multimedia Databases
Santiago, Chile, May 22, 2023
Root-join algorithm: first idea

- Building the groups
  - Select $\sqrt{n}$ random objects from the dataset as centers
  - Assign each object to the group of its closest center
    - If group is full, try with the second closest center
    - If that group is full too, try with the next closest center
    - Etc.

Complexity analysis (distance computations)
- Selecting the centers: 0 distance computations
- Forming the groups: $n\sqrt{n}$ distance computations
- Computing the 1-NN similarity join:
  - $\sqrt{n^2} = n$ distance computations for group
  - $\Rightarrow n\sqrt{n}$ distance computations in total
- Total cost: $0 + n\sqrt{n} + n\sqrt{n} = \Theta(n^{3/2})$ distance computations
Root-join algorithm: first idea

- Some observations
  - One could benefit from enlarging the groups
    - E.g., groups of $c\sqrt{n}$ objects
    - If $c$ is $\Theta(1)$, the total cost is $\Theta(n^{3/2})$ distance computations
    - This indeed improves the precision of the result
  - How to modify the algorithm for the k-NN self-similarity join?

Root-join algorithm

- Select $\sqrt{n}$ random objects as centers
- Distribute remaining objects on the groups
  - Groups have a maximum size of $c\sqrt{n}$
  - If the group where an object should be sorted into is already full, try with the next closest group
- For each object
  - Search k-NN within its group and the next closest group
    - Add more groups if necessary
Root-join algorithm

- Complexity analysis (distance computations)
  - Parameters $c$ and $k$ constants
  - Selecting the centers: 0 distance computations
  - Forming the groups: $n\sqrt{n}$ distance computations
  - Computing the k-NN similarity join:
    - Worst case for an object: $\max\{2c\sqrt{n}, k - 1 + c\sqrt{n}\} = \Theta(\sqrt{n})$ distance computations
    - Total cost: $0 + n\sqrt{n} + n\Theta(\sqrt{n}) = \Theta(n^{3/2})$ distance computations

Table 1

<table>
<thead>
<tr>
<th>Dataset</th>
<th>$c$</th>
<th>$k = 1$</th>
<th>$k = 4$</th>
<th>$k = 8$</th>
<th>$k = 16$</th>
</tr>
</thead>
<tbody>
<tr>
<td>DECAF</td>
<td>1</td>
<td>18.55%</td>
<td>17.22%</td>
<td>16.32%</td>
<td>15.48%</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>31.75%</td>
<td>30.26%</td>
<td>28.65%</td>
<td>27.25%</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>37.94%</td>
<td>35.69%</td>
<td>34.26%</td>
<td>32.38%</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>44.31%</td>
<td>41.41%</td>
<td>39.14%</td>
<td>38.19%</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>44.80%</td>
<td>41.93%</td>
<td>40.31%</td>
<td>37.91%</td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td>45.16%</td>
<td>41.92%</td>
<td>41.21%</td>
<td>38.49%</td>
</tr>
<tr>
<td>HOG</td>
<td>1</td>
<td>20.18%</td>
<td>18.90%</td>
<td>18.06%</td>
<td>17.16%</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>36.80%</td>
<td>34.32%</td>
<td>32.80%</td>
<td>31.10%</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>42.18%</td>
<td>39.53%</td>
<td>37.89%</td>
<td>35.29%</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>45.77%</td>
<td>42.77%</td>
<td>41.29%</td>
<td>38.94%</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>45.74%</td>
<td>42.68%</td>
<td>40.57%</td>
<td>38.81%</td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td>46.08%</td>
<td>42.88%</td>
<td>41.06%</td>
<td>39.03%</td>
</tr>
<tr>
<td>STRINGS</td>
<td>1</td>
<td>15.54%</td>
<td>15.86%</td>
<td>15.08%</td>
<td>13.24%</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>23.66%</td>
<td>24.55%</td>
<td>23.08%</td>
<td>20.11%</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>24.68%</td>
<td>26.26%</td>
<td>25.46%</td>
<td>22.56%</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>25.07%</td>
<td>25.87%</td>
<td>25.06%</td>
<td>22.92%</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>25.82%</td>
<td>26.71%</td>
<td>25.25%</td>
<td>22.45%</td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td>24.64%</td>
<td>26.92%</td>
<td>24.05%</td>
<td>22.16%</td>
</tr>
</tbody>
</table>

Similarity join and query languages

- How to formalize a similarity join in a query language
- Proposal by Ferrada et al. [FBH20]
  - Extend SPARQL with similarity join operator

Similarity join and query languages


AMW 2023 - Tutorial: Multimedia Databases
Santiago, Chile, May 22 2023
Final remarks

- Similarity concept is core to Multimedia Databases
- Many practical applications
  - Manufacturing industry
  - Computational science
  - Cultural heritage
  - Biometry
  - Pattern recognition
  - …

Final remarks

- Challenges
  - Efficient ways for computing similarity joins
    - Dynamic similarity models
    - How to optimize the query processing?
  - Non-metric distances
    - Indexing methods for this type of spaces?
  - New operators related with “similarity”
    - Reverse nearest neighbors
  - Multimodal data
Tutorial: Multimedia Databases

Prof. Benjamin Bustos

Millennium Institute Foundational Research on Data
Department of Computer Science
University of Chile