Multi-Dimensional Index Structures

Dr. Veit Köppen

OvG Universität Magdeburg
Fakultät für Informatik
Institut für Technische und Betriebliche Informationssysteme

2012
Overview

1. Distance Metrics
2. Classification of index structures
3. Exact index structures
4. Approximate index structures
Literature

Acknowledgements

- This presentation includes slides and contents provided by
  - Alexander Grebhahn
  - David Broneske

- Thanks!
Motivation

- Efficient retrieval of complex data, e.g., multimedia or Data Warehouse
- Multi-dimensional index structures required
- Challenge: *Curse of Dimensionality*
- Example: Increasing overlapping of MBRs of R-Tree
Query types revisited

- **Exact Match Query**: Identical point in database to the query
- **Range Query**: Every point in the query space (e.g., rectangle)
- **Nearest Neighbor (NN) Query**: Nearest point to the query
- **Approximate Nearest Neighbor Query**: A near point to the query

→ Results of an approx. NN query often $\neq$ NN query
→ Similarity measure required for neighbor relationship
**Similarity Measures**

- **Similarity Measure**: Function $s$ defining similarity between two data points
- $s = 1$ points are identical, $s = 0$ maximal dissimilarity
- Required for similarity queries e.g., NN-queries
- $s(x, y) \in [0, 1]$ is a similarity function if:
  
  \[
  \begin{align*}
  s(x, x) &= s(y, y) = 1 \quad \text{(identity)}, \\
  s(x, x) &\geq s(y, x), \quad \text{(minimality)} \\
  s(x, y) &= s(y, x) \quad \text{(symmetry)}, \\
  s(x, z) &\geq s(x, y) + s(y, z) - 1,
  \end{align*}
  \]
Distance Metrics

- Distance metrics can be used as similarity measures:
  - $d = 0$ points are identical
- $d(x, y)$ is a distance function if:
  - $d(x, y) \geq 0$ (non-negativity),
  - $d(x, y) = 0$ iff $x = y$ (identity),
  - $d(x, y) = d(y, x)$ (symmetry),
  - $d(x, z) \leq d(x, y) + d(y, z)$ (triangle inequality),
Distance Metric Functions

Different classes of distance metrics

- **Total distance metric**
  
  Minkowski Distance Metrics $L_p$:
  
  $$ L_p(x, y) = \left( \sum_{i=1}^{d} (|x_i - y_i|)^p \right)^{1/p} $$

- **Pseudo distance metric**: no non-negativity
- **Semi distance metric**: no triangle inequality
- **Semi pseudo distance metric**: no triangle inequality as well as non-negativity
  
  Dynamical Partial distance metric
Minkowski revisited

- Manhattan Distance Metric ($p = 1$)
  \[ L_1(x, y) = \sum_{i=1}^{d} |x_i - y_i| \]
- Most used: Euclidean Distance Metric ($p = 2$)
  \[ L_2(x, y) = \left( \sum_{i=1}^{d} (|x_i - y_i|)^2 \right)^{1/2} \]
- Computing square root $\rightarrow$ inaccuracy
- Supremum Distance Metric ($p = \infty$)
  \[ L_\infty(x, y) = \max_{i=1}^{d} \{|x_i - y_i|\} \]
Dynamical Partial

- Outliers in one dimension → strongly influence distance
- Limiting dimensions to $m$ (small), whereas data items have minimal distances

<table>
<thead>
<tr>
<th>Dimension:</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Item1:</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>Item2:</td>
<td>12</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

- For $m = 2$, dimensions 2 & 4 used for distance computation
Similarity Measure vs. Distance Metric

- Distance functions often used by limiting function to $[0, 1]$
- Nevertheless, distance functions quite restrictive w.r.t. psychological similarity perception
- Distance functions not necessarily unsuitable for similarity depending on application
Classification of Index Structures

Partitioning method

- **Space partitioning methods**: divide the whole space
- Example: VA-File

---

![Diagram of space partitioning methods](image)

- Vector File
- Approximation File

<table>
<thead>
<tr>
<th></th>
<th>Vector File</th>
<th>Approximation File</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.1 0.9</td>
<td>00 11</td>
</tr>
<tr>
<td>2</td>
<td>0.6 0.8</td>
<td>10 11</td>
</tr>
<tr>
<td>3</td>
<td>0.1 0.4</td>
<td>00 01</td>
</tr>
<tr>
<td>4</td>
<td>0.9 0.1</td>
<td>11 10</td>
</tr>
<tr>
<td>5</td>
<td>0.8 0.4</td>
<td>11 10</td>
</tr>
<tr>
<td>6</td>
<td>0.4 0.4</td>
<td>01 01</td>
</tr>
<tr>
<td>7</td>
<td>0.5 0.7</td>
<td>01 10</td>
</tr>
<tr>
<td>8</td>
<td>0.7 0.2</td>
<td>10 00</td>
</tr>
</tbody>
</table>
Partitioning method /2

- **Data partitioning methods**: divide necessary space to index every data item

![Diagram showing data partitioning methods](image-url)
Space vs. data partitioning

<table>
<thead>
<tr>
<th></th>
<th>Space partitioning</th>
<th>Data partitioning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Advantage</strong></td>
<td>easy partitioning algorithms</td>
<td>query on unpopulated space is not computed</td>
</tr>
<tr>
<td><strong>Disadvantage</strong></td>
<td>query on unpopulated space have to be computed</td>
<td>expensive partitioning algorithms and storage overhead</td>
</tr>
</tbody>
</table>
Quality of Results

- Exact index structures
  - Guarantee exact query results
  - Performance issue

- Approximate index structures
  - Not exact query result, but quite similar to exact result
  - E.g., $k$NN-Query: $k$ points quite similar to query point
  - Better performance
  - Performance $\leftrightarrow$ Precision
Exact-Index Structures

Covered here:

- R-Tree (1984)
- R+-Tree (1987)
- R*-Tree (1990)
- X-Tree (1996)
- SS-Tree (1996)
- SR-Tree (1997)
- VA-File (1997)
- VA+-File (2000)
- LPC-File (2002)
- Pyramid-Technique (1998)
- Ra*-Tree (1998)
- TV-Tree (1994)

adapted from Gaede & Günther (1998)
KNN-Algorithm (RKV-Algorithm)

- (Roussopoulos, Kelly, Vincent (1995))
- Algorithm applicable for all R-Tree derivates
- Differences: Computation of \textit{MinMaxDist} and \textit{MinDist}

```plaintext
1
2  RKV-PP(k, q, T) {
3  // k is the desired number of nearest neighbors, Q is a
4  // query point, and T is an R-tree
5  L := empty priority queue
6  Traverse(k, q, root[T], L)
7  return L
8
9 }
```
R-Tree III

```plaintext
KNearestTraversal(k, q, n, L) {  // n is an R-tree node
  if LeafNode(n) then
      for <M, O> in records[n] do // M is a MBR, O is a spatial object
        if Dist(q, M) < Max(L)
          then Insert(L, O)
      end if
  end for
  else ABL := Sort(records[n]) // Sort the records by MinDistance into an Active Branch List for <M, O> in ABL
    for <M, O> in ABL do
      if MinMaxDistance(q, M) < MAX(L) then // PROMISE-PRUNING
        Insert(L, Promise(MinMaxDistance(q, M)))
      end if
    end for
    for <M, O> in ABL do
      if MinDistance(q, M) < Max(L) then // K1 pruning
        if L contains a promise generated from M then
          Remove(L, Promis(MinMaxDistance(q, M)))
        end if
      end if
      KNearestTraversal(k, q, O, L)
    end if
  end if
}  
```
R-Tree IV

MinDist and MinMaxDist
R-Tree Drawback

Overlapping MBRs lead to performance leak
R⁺-Tree

R⁺-Tree: Sellis, Roussopoulos, and Faloutsos 1987

- Basic idea: Prohibition of overlappings → new insertion algorithm
- Requirement is hardly satisfiable → insertion may require adaptation of several leaves
- Adaptation implies partition in smaller MBRs without previous overflow → nodes with low load (i.e., with free capacity) → many nodes (degeneration)
- Spatial objects → possibly no encompassing MBR can be found → multiple entries necessary
**R⁺-Tree II**

**Insert Algorithm**

**Input:**
An R⁺-tree rooted at node R and an input rectangle IR

**Output:**
The new R⁺-tree that results after the insertion of IR

**Method:**
Find where IR should go and add it to the corresponding leaf nodes

1. [Search Intermediate Nodes]
   If R is not a leaf, then for each entry (p, RECT) of R check if RECT overlaps IR. If so, Insert(CHILD, IR), where CHILD is the node pointed to by p.

2. [Insert into Leaf Nodes]
   If R is a leaf, add IR in R. If after the new rectangle is inserted R has more than M entries, SplitNode(R) to re-organize the tree (see section 3.5).

adapted from Sellis et. al (1990)
R⁺-Tree III

Problems of R⁺-Tree:

- Objects have to be stored in several rectangle regions (clipping) – increased storage & modification effort

- Insert of objects possibly requires modification of several rectangle regions
**Problems of $R^+$-Tree /2:**

- Insert can lead to unavoidable partition of regions in certain situations

- Region modifications have consequences in both directions, to leaves as well as to the root

- Upper bound for entries in leaf nodes can not be guaranteed anymore
**R*-Tree**

**R*-Tree: Beckmann, Kriegel, Schneider, Seeger (1990)**

- Minimizing overlapping
- However, not forbidden
R*-Tree II

- Reinsertion of points
  - Delete points of the overfull MBR
  - If reinserting, points may be inserted into different MBRs
    → Avoids splits

- Yet another split algorithm considering
  - Overlap
  - Margin
  - Area
X-Tree: Berchthold, Keim, Kriegel (1996)

Idea based on two observations (regarding to the R-Tree):

1. Efficiency problems in high-dimensional space caused by increasing overlappings → sequential scan is more efficient than traversing the complete tree

2. Partitioning should take place in one predefined dimension (to minimize overlappings)
X-Tree II

Super Nodes

- Super node encompasses an arbitrary number of (DB) pages (no limitation in contrast to R-Tree)
- Sequential search within super node
- Super nodes are created dynamically ⇒ depends on degree of overlap

Conclusion: X-Tree is dynamic hybrid structure between one-dimensional array and R-Tree
X-Tree III

Graphically
Node Overflow

- Split history is maintained for each MBR ⇒ contains all dimensions already used for partitioning
  ⇒ MBR entry (2 points), splitted dimensions, pointers to subordinate node
- Overflow:
  1. Application of usual, topological partitioning
  2. Predefined degree of overlapping exceeded ⇒ selection of partitioning dimension based on split history
  3. Violation of balance ⇒ create new super node
TV-Tree

**TV-Tree: Lin, Jagadish, Faloutsos (1994)**

TV: Telescopic Vector

Basic Assumption: Ordering of dimensions. First dimensions are more important than others.

- Use only a few dimensions, use additional if necessary
- Objects with different dimensionality are comparable
- Compatible to all partitioning regions (e.g., MBRs/MBSs)
TV-Tree II

Adapted from Lin, Jagadish, Faloutsos (1994)
TV-Tree III

Advantages:
- Reduced comparisons compared to R-Tree
- Good storage utilization

Disadvantages:
- Splitting on reduced set of dimensions → high probability for huge amount of neighboring regions
SS-Tree

SS-Tree: White, Jain (1996)

- MBRs are described by two points
- Problems:
  - storage of two multidimensional points per MBR
  - Limited node size
  - For queries, comparison with both points of the MBR
- SS-Tree introduces Minimum Bounding Sphere (MBS)
- SS → Similarity Search
- MBS: center point + radius
SS-Tree II
SS-Tree III

MinDist and MinMaxDist
SS-Tree IV

Advantages:
- More MBSs per Node → reduced storage consumption
- Reduced amount of comparisons

Disadvantages:
- High volume of MBS → unpopulated space is indexed
- Overlapping regions
SR-Tree

**SR-Tree: Katayama, Satoh 1997**
Combination of R-Tree and SS-Tree

- MBR has a bigger diagonal than MBS $\rightarrow$ not optimal for kNN-Queries
- MBS has a bigger volume than MBR $\rightarrow$ unpopulated space is indexed
- SR-Tree uses the intersection of a MBR and a MBS
SR-Tree II

Köppen (FIN/ITI)  Multi-Dimensional Index Structures  2012  41 / 73
MinDist and MinMaxDist

- \[ \text{MinDist} = \max[\text{MinDist}(\text{MBR}), \text{MinDist}(\text{MBS})] \]
SR-Tree IV

Advantage:
- Improved NN-query performance

Disadvantages:
- Storage consumption
- Overlapping of resulting regions
- Computational effort at construction
A-Tree


Combination of R-Tree and VA-File

- Introduction of virtual bounding rectangles (VBR)
  - Approximates the diagonales of the parent and child MBRs
  - Approximation as bit-string
    → reduced comparison effort (compared to MBR)
A-Tree II
A-Tree III

```
V1  V2  V3
R1  V4  V5  V6
R2  V7  V8  V9
R3  V10 V11 V12
R4  V... R5  V...
R6  V... R7  V...
R8  V... R9  VA  VB  VC
R10 V...
R11 V...
R12 V...
A  B  C
```
A-Tree IV

Advantage:
- Knowledge about the underlying partitioning of the MBR without loading

Disadvantage:
- Overlapping of MBRs and VBRs
Ra*-Tree

Ra*-Tree: Jürgens, Lenz (1998)

- Support for OLAP
- Predominantly used: Aggregate functions
  → SUM, MAX, AVG, COUNT, ...
- Ra*-Tree stores chosen aggregated values of underlying leaves in every inner node (materialized views)
  → a stands for aggregated
$R_a^*$-Tree II

$R_a^*$-Tree caching COUNT and SUM
$R_a^*$-Tree

Advantage:
- Fast evaluation of aggregations

Disadvantage:
- Storage overhead
  - Tree height increases
R-Tree Family - Conclusion

Many different improvements of R-Tree

- Reduced overlapping
- Different Regions
- Minimizing used dimensions

→ Improved performance for chosen query types
→ Better storage utilization

But still:

- Overlapping of Regions → visiting of multiple paths when executing queries
VA-File

VA-File revisited

- User defined number of bits per dimension
- Equally filled cells
- Approximation File & Vector File

![Diagram of VA-File with user-defined number of bits per dimension and equally filled cells. The diagram includes a grid with letters A to Z, and numbers 00 to 11, representing the different cells and their corresponding bit values.]

Approximation File

<table>
<thead>
<tr>
<th>Letter</th>
<th>Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>00 11</td>
</tr>
<tr>
<td>B</td>
<td>01 11</td>
</tr>
<tr>
<td>C</td>
<td>01 11</td>
</tr>
<tr>
<td>D</td>
<td>10 11</td>
</tr>
<tr>
<td>E</td>
<td>01 10</td>
</tr>
<tr>
<td>F</td>
<td>01 10</td>
</tr>
<tr>
<td>G</td>
<td>01 10</td>
</tr>
<tr>
<td>H</td>
<td>00 10</td>
</tr>
<tr>
<td>I</td>
<td>00 10</td>
</tr>
<tr>
<td>J</td>
<td>10 11</td>
</tr>
<tr>
<td>K</td>
<td>11 11</td>
</tr>
<tr>
<td>L</td>
<td>11 10</td>
</tr>
<tr>
<td>M</td>
<td>11 10</td>
</tr>
<tr>
<td>N</td>
<td>11 01</td>
</tr>
<tr>
<td>O</td>
<td>11 01</td>
</tr>
<tr>
<td>P</td>
<td>11 00</td>
</tr>
<tr>
<td>Q</td>
<td>10 01</td>
</tr>
<tr>
<td>R</td>
<td>11 00</td>
</tr>
<tr>
<td>S</td>
<td>10 00</td>
</tr>
<tr>
<td>T</td>
<td>01 00</td>
</tr>
<tr>
<td>U</td>
<td>10 00</td>
</tr>
<tr>
<td>V</td>
<td>10 01</td>
</tr>
<tr>
<td>W</td>
<td>10 10</td>
</tr>
<tr>
<td>X</td>
<td>00 01</td>
</tr>
<tr>
<td>Y</td>
<td>01 01</td>
</tr>
<tr>
<td>Z</td>
<td>00 00</td>
</tr>
</tbody>
</table>
VA⁺-File


VA-File allocates fixed number of bits for each dimension
- Some dimensions have more importance (cf. TV-Tree)
- VA⁺-File uses dynamical amount of bits for dimension
- Karhunen-Loève Transformation: eliminate correlations between dimensions
- Definition of the cellborders using Lloyds Algorithm (Clustering algorithm)
VA⁺-File

Advantage:
- Reduction of candidate size
→ Lower I/O costs

Disadvantage:
- High construction time
- Higher computational overhead
LPC-File


- Basis: VA-File, with equal length cells
- Additional bits for polar coordinates:
  - Angle from diagonal
  - Radius from left-lower corner
LPC-File II

- More accurate description of candidate range
  → Reduced candidate set
LPC-File III

Advantage:
- Minimizing candidate set
  → Query performance increased

Disadvantage:
- Computational Overhead
- Storage Overhead - insignificant for increasing dimensions
VA-File Family Conclusion

Good approach minimizing Advantage:

- Sequential access of approximation vectors
  → Minimizing I/O costs
- Extends sequential scan: Every query type executable

Disadvantage:

- Computational complexity increasing with number of points
Pyramid Technique

Berchtold, Böhm, Kriegel (1998)

- Space partitioning index structure
- Partition of d-dimensional space in $2 \cdot d$ pyramids

$$i = \begin{cases} j_{\text{max}} & \text{if } x_{j_{\text{max}}} < 0.5 \\ (j_{\text{max}} + d) & \text{if } x_{j_{\text{max}}} \geq 0.5 \end{cases}$$

- Partitioning of pyramids for managing space
  - Different partitions needed for different kinds of queries
Pyramid Technique II

Berchtold, Böhm, Kriegel (1998) Supporting range queries
Pyramid Technique III

Lee, Kim (2003) Supporting kNN queries
Pyramid Technique VI

KNN-Algorithm

```plaintext
for i = 0 to 2d-1 do
    dist = MINDIST(q,spi);
    ENQUEUE(queue, spi, dist);
end for
while not ISEMPTY(queue) do
    Element = DEQUEUE(queue);
    if Element is a spherical pyramid then
        for each bounding slice in a spherical pyramid do
            dist = MINDIST(q,BSI);
            ENQUEUE(queue, BSI, dist);
        end for
    else if Element is a bounding slice then
        for each object in a bounding slice do
            dist = DIST_QUERY_TO_OBJ(q, object);
            ENQUEUE(queue, object, dist);
        end for
    else /* Element is an object */
        report element as the next nearest object
    end if
end while
```
Pyramid Technique V

Advantage:

- Less comparisons

Disadvantage:

- Different partitioning for different query types
Approximation based Index Structures

Quality of Results

- Performance intensive computation of exact query result
- Often: nearly exact results sufficient
- Precision vs. Performance
Locality Sensitive Hashing

Basic Idea

- Instead of scattering the data over buckets, ordering similar data items into same bucket with high probability
  - Preserves neighbor relationships

- \((P_1, P_2, r, cr)\)-sensitive hash functions \(h\) needed, so that:
  - if \(|p - q| < r\), then \(Pr[h(p) = h(q)] > P_1\)
  - if \(|p - q| > cr\), then \(Pr[h(p) = h(q)] < P_2\)
  - \(P_1 \gg P_2\)
P-stable LSH

- Usage of $p$-stable distributions

$\mathcal{D}$ is $p$-stable if for $n$ real numbers $v_1, \ldots, v_n$ and independent and identically distributed random variables $X_1, \ldots, X_n$ and $X \in \mathcal{D}$, $p \geq 0$:

$$
\sum_{i=1}^{n} (v_i X_i) \text{ distributed as } (\sum_{i=1}^{n} (\|v_i\|^p))^{1/p} X
$$
P-stable LSH II

- Chosing a $d$-dimensional vector $\vec{a}$ consisting of $d$ random variables from a $p$-stable distribution
  - The dot product of $\vec{a}$ and data item $\vec{v}$ is well-known distributed
  - $\vec{a} \cdot \vec{v} \sim \left( \sum_{i=1}^{d} (\|v_i\|^p) \right)^{1/p} X$

- Several such dot products result in an approximation of $\|\vec{v}\|_p$

- Approximation of $L_p$ distance metric
  - Manhattan Distance Metric: Standard Cauchy Distribution
    $c(x) = \frac{1}{\pi (1+x^2)}$, $p = 1$
  - Euclidean Distance Metric: Standard Gaussian Distribution
    $g(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}$, $p = 2$
P-stable LSH III

Partitioning

Legend
- - - bucket borders of function $h_1$
- - - bucket borders of function $h_2$

KNN-Search:
- Retrieve candidates from every bucket, the query falls into
- Refine candidates to k nearest points
Prototype based approach

Basic idea

- Usage of randomly chosen data points for indexing the space → prototypes
- Classifying points according to their distance to prototypes
Prototype based approach

Basic idea

- Usage of randomly chosen data points for indexing the space → prototypes
- Classifying points according to their distance to prototypes
Prototype based approach

Basic idea

- Usage of randomly chosen data points for indexing the space → prototypes
- Classifying points according to their distance to prototypes
Prototype based approach

Basic idea

- Usage of randomly chosen data points for indexing the space \rightarrow prototypes
- Classifying points according to their distance to prototypes

![Diagram showing prototype-based approach with points P1, P2, and P3 and their respective regions]

Köppen (FIN/ITI)
Multi-Dimensional Index Structures
Prototype based approach II

Conclusion

- Number of comparisons dependent on amount of prototypes
- Can be used to assign hash values or for optimized sequential search
- Perfect amount of prototypes not determinable
- Different positions of prototypes lead to different precisions
Approximate vs. Exact Index Structures

<table>
<thead>
<tr>
<th>Approximate</th>
<th>Exact</th>
</tr>
</thead>
<tbody>
<tr>
<td>Advantage</td>
<td>Good query performance</td>
</tr>
<tr>
<td>Disadvantage</td>
<td>No exact result</td>
</tr>
<tr>
<td></td>
<td>Precision issues</td>
</tr>
</tbody>
</table>

- Different query types:
  - Approximate Nearest Neighbor vs. Nearest Neighbor
  - Approximate Range Queries?
Thank you for your attention!