13 Indexes for Multimedia Data

13.0 Indexes for Multimedia Data

- Speed up search through indexing
  - Efficient management of multidimensional information
    - Pre-structuring of data for the subsequent search functionality
    - Efficient data structures, combined with search and comparison algorithms
  - Transition from set semantics to list semantics
    - To which degree does the object from the database satisfy the query?

13.0 Indexes for Multimedia Data

- Requirements for a multidimensional index structure
  - Correctness and completeness of the corresponding indexing algorithms
  - Scalability with dimension growth
  - Support objects which are not real-valued vectors
  - Search efficiency (sublinear)

13.0 Indexes for Multimedia Data

- Multi-dimensional data
  - Images
  - Audio data
  - Video data

- Description of multimedia objects
  - Usually (multidimensional) real-valued feature vectors
  - But also: skeletons, chain codes, ...

- The sequential search for similar objects in databases is very inefficient

- How can we speed up the search?
13.0 Indexes for Multimedia Data

- **Fundamental problem:**
  - The more dimensions, the more comparisons are needed
  - There is currently no truly scalable indexing
  - **Cause:** “Curse of Dimensionality” (Richard Bellman)
    - The volume of space grows exponentially with the number of its dimensions

13.0 Query Types

- **Exact search**
  - Point search
  - Area search
- **k-nearest-neighbor search** (k - NN-search)
  - Find the k objects that have the least distance to the object given as reference in the request
  - k-NN search is usually only calculated on approximation basis (with a specified error) due to the high cost
- **Reverse-nearest-neighbor search**
  - Find all the objects whose nearest neighbor is provided in the query

13.0 Tree Structures

- **Search in database systems**
  - B-tree structures allow exact search with logarithmic costs

- **Search in multimedia databases**
  - The data is multidimensional, B-trees however, support only one-dimensional search
- Are there any possibilities to extend tree functionality for multidimensional data?

13.0 Tree Structures

- **The basic idea** of multidimensional trees
  - Describe the sets of points through geometric regions, which comprise the points (clusters)
  - The clusters are considered for the actual search and not the individual points
  - Clusters can contain each other, resulting in a hierarchical structure

- **Differentiating criterias for tree structures:**
  - **Cluster construction:**
    - Completely fragmenting the space or
    - Grouping data locally
  - **Cluster overlap:**
    - Overlapping or
    - Disjoint
  - **Balance:**
    - Balanced or
    - Unbalanced
13.0 Tree Structures

- **Object storage:**
  - Objects in leaves and nodes, or
  - Objects only in the leaves
- **Geometry:**
  - Hyper-spheres,
  - Hyper-cube,
  - ...

Each internal node has between –

13.1 R-Trees

- The **R-tree** (Guttman, 1984) is the prototype of a multi-dimensional extension of the classical B-trees
- Frequently used for low-dimensional applications (used to about 10 dimensions), such as geographic information systems
- More scalable versions: R*-Trees, R*-Trees and X-Trees (each up to 20 dimensions for uniform distributed data)

13.1 R-Tree Structure

- **Dynamic Index Structure**
  (insert, update and delete are possible)
- Data structure
  - **Data pages** are leaf nodes and store clustered point data and data objects
  - **Directory pages** are the internal nodes and store directory entries
  - Multidimensional data are structured with the help of Minimum Bounding Rectangles (MBRs)

13.1 R-Tree Example

13.1 R-Tree Characteristics

- Local grouping for clustering
- Overlapping clusters (the more the clusters overlap the more inefficient is the index)
- Height balanced tree structure (therefore all the children of a node in the tree have about the same number of successors)
- Objects are stored, only in the leaves
  - Internal nodes are used for navigation
- MBRs are used as a geometry

13.1 R-Tree Properties

- The root has at least two children
- Each internal node has between m and M children
- M and m ≤ M / 2 are pre-defined parameters
- For each entry (I, child-pointer) in an internal node, I is the smallest rectangle that contains the rectangles of the child nodes
### 13.1 R-Tree Properties

- For each index entry \((I, \text{tuple-id})\) in a leaf, \(I\) is the smallest bounding rectangle that contains the data object (with the ID tuple-id)
- All the leaves in the tree are on the same level
- All leaves have between \(m\) and \(M\) index records

### 13.1 Operations of R-Trees

- The essential operations for the use and management of an R-tree are
  - Search
  - Insert
  - Updates
  - Delete
  - Splitting

### 13.1 Searching in R-Trees

- The tree is searched **recursively** from the root to the leaves
  - One path is selected
  - If the requested record has not been found in that sub-tree, the next path is traversed
- The path selection is arbitrary

### 13.1 Search Algorithm

- All the index entries which intersect with the **search rectangle** \(S\) are traversed
  - The search in internal nodes
    - Check each object for intersection with \(S\)
    - For all intersecting entries continue the search in their children
  - The search in leaf nodes
    - Check all the entries to determine whether they intersect \(S\)
    - Take all the correct objects in the result set

### 13.1 Example

- Check only 7 nodes instead of 12
13.1 Insert

- **Procedure**
  - The best leaf page is chosen (ChooseLeaf) considering the spatial criteria
    - Beast leaf: the leaf that needs the smallest volume growth to include the new object
  - The object will be inserted there if there is enough room (number of objects in the node < M)

13.1 Insert

- If there is no more place left in the node, it is considered a case for overflow and the node is divided (SplitNode)
  - Goal of the split is to result in minimal overlap and as small dead space as possible
  - Interval of the parent node must be adapted to the new object (AdjustTree)
  - If the root is reached by division, then create a new root whose children are the two split nodes of the old root

### 13.1 R-Tree Insert Example

![R-Tree Insert Example Diagram]

- Inserting P either in R7 or R9
- In R7, it needs more space, but does not overlap

### 13.1 Heuristics

- An object is always inserted in the nodes, to which it produces the smallest increase in volume
- If it falls in the interior of a MBR no enlargement is need
- If there are several possible nodes, then select the one with the **smallest volume**

### 13.1 Insert with Overflow

![Insert with Overflow Diagram]

- If an object is inserted in a full node, then the M+1 objects will be divided among two new nodes
- The goal in splitting is that it should rarely be needed to traverse both resulting nodes on subsequent searches
  - Therefore use small MBRs. This leads to minimal overlapping with other MBRs

### 13.1 SplitNode

- If an object is inserted in a full node, then the M+1 objects will be divided among two new nodes
- The goal in splitting is that it should rarely be needed to traverse both resulting nodes on subsequent searches
  - Therefore use small MBRs. This leads to minimal overlapping with other MBRs
### 13.1 Split Example

- Calculate the minimum total area of two rectangles, and minimize the dead space

![Bad split vs Better split](image)

### 13.1 Overflow Problem

- Deciding on how exactly to perform the splits is **not trivial**
  - All objects of the old MBR can be divided in different ways on two new MBRs
  - The volume of both resulting MBRs should remain as small as possible
  - The naive approach of checking checks all splits and calculate the resulting volumes is not possible

- **Two approaches**
  - With **quadratic cost**
  - With **linear cost**

### 13.1 Overflow Problem

- **Procedure with quadratic cost**
  - Compute for each 2 objects the necessary MBR and choose the pair with the largest MBR
  - Since these two objects should not occur in an MBR, they will be used as **starting points** for two new MBRs
  - Compute for all other objects, the difference of the necessary volume increase with respect to both MBRs

- **Procedure with linear cost**
  - In each dimension:
    - Find the rectangle with the highest minimum coordinates, and the rectangle with the **smallest maximum coordinates**
    - Determine the distance between these two coordinates, and normalize it on the size of all the rectangles in this dimension
  - Determine the two starting points of the new MBRs as the two objects with the highest normalized distance

### 13.1 Example

- **x-direction:** select A and E, as \( d_x = \text{diff}_x / \text{max}_x = 5 / 14 \)
  - Since \( d_x < d_y \), C and D are chosen for the split
- **y-direction:** select C and D, as \( d_y = \text{diff}_y / \text{max}_y = 8 / 13 \)
13.1 Overflow Problem

- Classify all remaining objects the MBR with the smallest volume growth
- The linear process is a simplification of the quadratic method
- It is usually sufficient providing similar quality of the split (minimal overlap of the resulting MBRs)

13.1 Delete

- Procedure
  - Search the leaf node with the object to delete (FindLeaf)
  - Delete the object (deleteRecord)
  - The tree is condensed (CondenseTree) if the resulting node has < m objects
  - When condensing, a node is completely erased and the objects of the node which should have remained are reinserted
  - If the root remains with just one child, the child will become the new root

13.1 Example

- An object from R9 is deleted (1 object remains in R9, but m = 2)
  - Due to few objects R9 is deleted, and R2 is reduced (condenseTree)

13.1 Update

- If a record is updated, its surrounding rectangle can change
- The index entry must then be deleted updated and then re-inserted

13.1 Improved Versions of R-Trees

- Where are R-trees inefficient?
  - They allow overlapping between neighboring MBRs
- R*-Trees (Sellis ua, 1987)
  - Overlapping of neighboring MBRs are prohibited
  - This may lead to identical leafs occurring more than once in the tree
  - Improve search efficiency, but similar scalability as R-trees
13.1 R*-Trees

- Overlaps are not permitted (A and P)
- Data rectangles are divided and may be present (e.g., G) in several leafs

13.1 Operations in R*-Trees

- Differences to the R-tree
  - Insert
    - Data object can be inserted into several leafs
    - Splitting continues downwards, since no overlaps are allowed
  - Delete
    - There is no more minimum number of children

13.1 Performance

- The main advantage of R*-trees is to improve the search performance
- Especially for point queries, this saves 50% of access time
- Drawback is the low occupancy of nodes resulting through many splits
- R*-trees often degenerate with the increasing number of changes

13.1 More Versions

- R*-trees and X-trees improve the performance of the R*-trees (Kriegel and others, 1990/1996)
  - Improved split algorithm in R*-trees
  - “Extended nodes” in X-trees allow sequential search of larger objects
  - Scalable up to 20 dimensions

13.2 M-Trees

- M-tree (Ciaccia et al, 1997) allows the use of arbitrary metrics for comparison of objects ("metric trees")
  - R-trees only work with Euclidean metrics, but what about for example, the editing distance?
  - Use the triangle inequality to check sub-trees
  - Geometry is determined by the distance function

13.2 Metric Space

- A metric space is a pair of $M = (U, d)$
  - $U$ is the universe of all possible values
  - $d$ is a metric
  - For all $x, y, z \in U$:
    - $d(x, y) \geq 0, d(x, y) = 0$ iff $x = y$
    - $d(x, y) = d(y, x)$
    - $d(x, y) \leq d(x, z) + d(z, y)$ (triangle inequality)
13.2 Triangle Inequality

- **Precomputed:**
  Distances for all pairs of points
- **Task:** Find the object with the smallest distance to Q
- Distance between Q and a is 2
- Distance between Q and b is 7.81
- Can C be the best object?
- \( d(Q, b) \leq d(Q, c) + d(b, c) \)
- \( 5.51 \leq d(Q, c) \)
- No. Therefore a is better

13.2 Partitioning

- The M-tree partitions the objects in \( \varepsilon \)-environments with certain radius

### Triangle Inequality

![Triangle Inequality Diagram](image)

### Partitioning

![Partitioning Diagram](image)

13.2 M-Trees

- M-trees are similar to R-trees, but use the distance information

![M-Trees Diagram](image)

- **Internal nodes** have
  - A routing object
  - The radius of their region and
  - A distance to the parent node
- **Leaf nodes** have
  - The values of the indexed objects and
  - Their distance from the parent node

![M-Trees Diagram](image)

- Precomputed distances to the respective parent nodes allow fast searching (“fast pruning”)
- \( d(v_P, v_N) \) is precomputed. We don’t need \( d(q, v_N) \)
  if \( |d(q, v_P) - d(v_P, v_N)| > r_N + r \)
13.2 M-Trees

- **Insert** is performed as by R-trees with the smallest expansion of the region radius
- At overflow, a **split** is performed
  - No volumes are however calculated (as in MBRs in the R-tree)
  - Delete the node and choose two new routing objects
  - **Heuristic**: Minimize the maximum of the two resulting region radiiuses
  - Attribute then the routing objects to the new regions alternating between their nearest neighbors (Balanced Split)

- M-Trees overview
  - Allow a variety of **distance functions**
  - Use triangle inequality for pruning
  - The dimensionality is also very limited

Next lecture

- Indexes for Multimedia Data
  - Curse of Dimensionality
  - Dimension Reduction
  - GEMINI Indexing