Spatiotemporal Access to Moving Objects

Hao LIU, Xu GENG
17/04/2018
Contents

• Overview & applications
• Spatiotemporal queries
• Moving objects modeling
  • Sampled locations
  • Linear function of time
• Indexing structure
  • TPR-tree
  • Tree organization heuristics
  • Operations
<table>
<thead>
<tr>
<th>Spatiotemporal vs. Spatial</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Space and time</td>
</tr>
<tr>
<td>• Moving objects</td>
</tr>
<tr>
<td>• Frequent updates</td>
</tr>
<tr>
<td>• Space</td>
</tr>
<tr>
<td>• Static objects</td>
</tr>
<tr>
<td>• Less frequent updates</td>
</tr>
</tbody>
</table>
Application: Air Traffic Controller

- Moving objects: airplanes
- Example: which airplanes will probably arrive at the airport in the next 5 minutes?

(Euclidean space: straight and direct Euclidean distance as the distance)
Application: Taxi-hailing Service

• Moving objects: taxis on the city roads

• Example: what are the nearest five taxis at this moment?
Application: Wireless Communication System

• Moving objects: mobile phone users

• Example: how many mobile phone users have been in area #47 in last 2 hours?

(Cellular network)
Location-based Spatiotemporal Queries

• Spatial queries at a certain timestamp or within a certain time interval
  • Intersection join
  • Window query
  • kNN
  • ...

Return the 2 nearest neighbors of q at time $T_1$ and $T_2$
Trajectory-based Spatiotemporal Queries

- Trajectory: path that a moving object follows over a time period
- Topological query
  - Enter
  - Leave
  - Cross
  - Bypass

$T_1 \rightarrow T_2$
Trajectory-based Spatiotemporal Queries

- Navigational query
  - Heading
  - Speed
  - Distance
  - Area

- Return the moving direction of $p_1$
- Return the objects that have moved to the north
- Return the travel speed of $p_2$
- Return the objects having traveled at speed of no less than 20 m/min
Aggregate Spatiotemporal Queries

• Aggregate query: summarized information about moving objects that lie in a query region during a query interval

Return the number of objects that have visited \( W \) between time \( T_1 \) and \( T_2 \)

\( T_1 \rightarrow T_2 \)
Modeling Moving Objects

• Historical queries: by sampled locations
  • Historical R-tree
  • 3D R-tree

• Predictive queries: by linear function of time
  • TPR tree
R-tree

- Spatial indexing with MBR (Minimum Bounding Rectangle)
- How to deal with objects that evolves?
R-tree Storing All Previous States?
Historical R-tree
3D R-tree

• Time is viewed as another dimension

• A trajectory of a point in 2D space is transformed to a set of 3D line segments, bounded by MBB (Minimum Bounding Box), its 2D projection reflects the origin trajectory

• 2D topological queries are transformed to 3D static range queries
Historical R-tree vs. 3D R-tree

**Historical H-tree**
- **Advantages**
  - All the historical states of the moving objects are maintained with reduced disk space
  - Efficient access to the previous states
- **Disadvantages**
  - Inefficient in preserving trajectory of objects

**3D R-tree**
- **Advantages**
  - Good at processing trajectory-based queries
- **Disadvantages**
  - There can be large dead spaces in an MBB, so overlapping is high in the whole index structure
Modeling Moving Objects

• Historical queries: by sampled locations
  • Historical R-tree
  • 3D R-tree

• Predictive queries: by linear function of time
  • TPR tree
Moving objects

• Assume linear motion, modeling position as a function of time
  • $x(t) = x(t_{ref}) + v(t - t_{ref})$
  • Make tentative future predictions
  • Avoid frequent update

• What to store
  • Set $t_{ref}$ as index creation time
  • Store $(x(t_{ref}), v)$ for tracking moving objects
MBR for tracking moving objects

Fig 1: Initial setting for reference position and velocity at $t_{\text{ref}}$

Fig 2: MBR under initial setting

Fig 3: Position at time $t$. The original MBR assignment is deteriorated.

Fig 4: The ideal MBR assignment in $t$. $T = t_{\text{ref}}$
TPR tree – index structure

- **Time Parameterized R tree**
  - Design for querying moving objects in a period of time *in the future*
  - Leaf nodes
    - Position of moving object
      - Represented by \((x_{ref}, v)\)
    - Pointer to moving object
  - Internal (non-leaf) nodes
    - Bounding rectangle
      - Represented by \((\text{MBR}, \text{VBR})\)
    - Pointer to subtree
Example in tracking moving rectangles

VBRs: \{left, right, bottom, top\}

\[ a_v = \{1,1,1,1\}; \quad b_v = \{-2,-2,-2,-2\}; \quad N_{1v} = \{-2,1,-2,1\} \]

\[ c_v = \{-2,0,0,2\}; \quad d_v = \{-1,-1,1,1\}; \quad N_{2v} = \{-2,0,0,2\} \]

- Some characteristics on this bounding strategy
  - The bounding strategy is conservative – keeps expanding
  - Avoids excessive storage cost.
  - Bound rectangle tightened when new rectangle inserted or deleted
Heuristics on designing tree

• The TPR tree is designed for timestamp queries in $[T_c, T_c + H]$
  • $T_c$: Current update time
  • $H$: Tree parameter – the timespan the tree can see in the future

• TPR-tree
  • Given an objective function $A(t = T_c)$ for static data
  • Static indexing structures minimize $A(t = T_c)$ during tree organization
  • TPR-tree minimize the integral over time:

$$\text{minimize } \int_{T_c}^{T_c+H} A(t)dt$$
Time as a parameter

• Left: 
  \[ \text{minimize } A(t = T_c) \]

• Right: 
  \[ \text{minimize } \int_{T_c}^{T_c+H} A(t)dt \]
Objective

• Use objective from R*-tree
  • The area of a bounding rectangle
  • The overlap of two rectangle.
  • The perimeter of a bound rectangle
  • The distance between the centroids

• The four objectives are adapted in different parts of R*-tree and TPR-tree algorithm
  • Keeps bound rectangles small
  • The probability of rectangle intersects query region is small

For TPR-tree

\[ \int_{Tc}^{Tc+H} A(N, t)dt \] [1]
\[ \int_{Tc}^{Tc+H} OVR(N_1, N_2, t)dt \] [2]
\[ \int_{Tc}^{Tc+H} P(N, t)dt \] [3]
\[ \int_{Tc}^{Tc+H} CDIST(N_1, N_2, t)dt \] [4]
**Insertion -- ChooseSubTree**

**R-tree**
- 1. \( n = \text{root} \)
- 2. **IF** \( n \) is a leaf
  - return \( n \)
  - **ELSE**
    - choose entry that minimize MBR area ([1])
- 3. \( n = \) chosen entry, go to 2

**R*-tree and TPR-tree**
- 1. \( n = \text{root} \)
- 2. **IF** \( n \) is a leaf
  - return \( n \)
  - **ELSE**
    - choose entry that minimize MBR area ([1]); resolve ties by smallest MBR size
- 3. \( n = \) chosen entry, go to 2
Insertion -- ChooseSubTree

• Try to insert k:
  • Choose N5
    • min BR area increment
  • Choose N1
    • Minimize BR overlap

R*-tree and TPR-tree
• 1. n=root
• 2. IF n is a leaf
   return n
   IF n’s child is leaf
   Choose entry that minimize BR overlap ([2]) with siblings; resolve ties by smallest area entanglement
   ELIF n’s child is non-leaf
   Choose entry that minimize MBR area ([1]); resolve ties by smallest MBR size
• 3. n = chosen entry, go to 2
Re-insert

- When a leaf node is full
- Remove and re-insert a fraction of entries
  - Select entries with largest centroid distance
  - If still full, do split

- N1 is full and k is awaiting:
  - Re-insert b
    - Largest centroid distance CDIST(b,N1), compared to a,c,k
    - B is re-inserted to N1
    - Do split
Split

Step 1 ChooseSplitAxis

• For each axis (x and y in this case):
  • 1. Sort entries by lower value of the rectangle (a,k,c,b)
  • 2. Determine all possible entry allocations (1-3, 2-2, 3-1)
  • 3. Compute $S = \text{sum of perimeter for each allocation}$
  • 4. go to 1 and do the same for higher value, accumulate $S$
  • 5.6...

• Choose the split axis with minimum $S$
Split

Step 2 ChooseSplitIndex

- Minimize overlap between MBRs
  - Suppose x-axis is chosen in ChooseSplitAxis
  - Considering 3 different divisions
    - 2-2 division has minimal overlap
    - Split using 2-2 division

Insertion result
Deletion

• Identify the leaf node contains the entry to be deleted and remove the entry
• If leaf node underflows
  • Re-insert all entries of the node
• Else
  • Remove entry and terminate
• Propagate to upper levels if needed

• Bound rectangles are tightened after insertion or deletion
Reference


Reference


