VoR-Tree: R-trees with Voronoi Diagrams for Efficient Processing of Spatial Nearest Neighbor Queries

Instructor: Cyrus Shahabi
Outline

• Introduction
  – Motivation: I/O-efficient spatial query processing
• Our Index Structure: VoR-tree
  – Voronoi Diagram
  – R-tree
  – VoR-tree
• Query Processing Using VoR-tree
  – Related works
  – k Nearest Neighbor Query
  – k Aggregate Nearest Neighbor Query
  – Reverse k Nearest Neighbor Query
• Performance Evaluation
• Summary and Future Directions
Motivation

- **Index-based processing of Nearest Neighbor queries**
  - Spatial index provides fast access by hierarchical grouping
  - Algorithms utilize aggregate information to *minimize I/O* operations

---

**Search Region of p:**
- A possible better result must be inside this region

---

**Step 1:**
- Filtering through iterative pruning
  - R-trees

**Step 2:**
- Refinement through exploration
  - R-trees
Motivation

- **Index-based processing of Nearest Neighbor queries**

Traverse along edges of Delaunay graph to minimize/maximize a function \( f \) …

**Search Region of \( p \):** a possible better result must be inside this region

- **Step 1:**
  - Filtering through iterative pruning
    - R-trees

- **Step 2:**
  - Refinement through exploration
    - Voronoi diagrams
Voronoi Diagrams

- Given a set of spatial objects, a Voronoi diagram *uniquely* partitions the space into disjoint regions (cells).
- The region including object \( p \) includes all locations which are closer to \( p \) than to any other object \( p' \).

**Ordinary Voronoi Diagram**

**Dataset:**
- Points

**Distance \( D(.,.) \):**
- Euclidean (\( L_2 \))

**Voronoi Cell of \( p \)**

**Voronoi Neighbors of \( p \)**

Point \( q \) inside the cell of \( p \)

\[ D(q, p) \leq D(q, p') \]
R-tree: Classic Spatial Index Structure

- Hierarchical grouping of objects into MBRs
- The best NN query processing algorithms utilize R-tree
- Algorithms utilize mindist()
VoR-tree = R-tree + Voronoi Diagram

- We incorporate Voronoi diagram into R-tree → VoR-tree
- Voronoi records are stored with the data of each point
- All R-tree-based algorithms are still applicable using VoR-tree
- VoR-tree facilitates exploring the space (e.g., $p_4$-$p_{11}$)

$VN(p_4) = \{ p_5, p_6, p_{12}, p_{14}, p_8, p_7 \}$
$VN(p_4) = \{ a, b, c, d, e, f \}$

Voronoi Record of $p_4$
Query Processing using VoR-tree

👍 I/O-efficient query processing

- Use the information provided in VoR-tree to find the result with the least number of I/O operations
- When a candidate result \( p \) is found, examine only the points inside the search region of \( p \)

👎 Disk space overhead -> ok for enterprise applications
Related Work

• **k Nearest Neighbor (kNN)**
  – Roussopoulos et al., SIGMOD’95
  – Korn et al., VLDB’96
  – Cheung et al., SIGMOD Record, 1998
  – Seidl et al., SIGMOD’98
  – Hjaltson et al., TODS 42(2), 1999
  – Jung et al., IEEE TKDE 2002

• **Reverse k Nearest Neighbor (RkNN)**
  – Korn et al., SIGMOD’00
  – Yang et al., ICDE’01
  – Stanoi et al., VLDB’01
  – Benetis et al., VLDB Journal, vol. 15, 2006
  – Tao et al., VLDB’04
  – Wu et al., VLDB’08

• **k Aggregate Nearest Neighbor (kANN)**
  – Papadias et al., ICDE’04
  – Papadias et al., TODS 30(2), 2005

• **Spatial Skyline**
  – Borzsonyi et al., ICDE’01
  – Tan et al., VLDB’01
  – Kossmann et al., VLDB’02
  – Chomicki et al., ICDE’03
  – Papadias et al., SIGMOD’03
  – Sharifzadeh et al., VLDB’06, TODS’09
**kNN: k Nearest Neighbor Query**

- Given: point $q$ and int $k$
- Goal: find the $k$ closest data points to $q$; $k$ points $p_i$ in $P$ where $D(q,p_i) \leq D(q,p)$ for all points $p$ in $P \setminus \{p_1, \ldots, p_k\}$
- R-tree-based Algorithm:
  BFS [Hjaltson et al., TODS 1999]
- Our VoR-tree-based Algorithm:
  VR-kNN

![Diagram of kNN query with point $q$ and set of points $P$. $3NN(q) = \{p_1, p_2, p_3\}$]
VR-1NN: step 1

\[ \text{mindist}(N, q) = \text{Lower bound on the distance between } q \text{ and any point in } N \]
VR-1NN: step 1

minheap H

VLDB 2010 (Singapore)
VR-1NN: step 1

minheap H

VLDB 2010 (Singapore)
VR-1NN: step 1

candidate 1\textsuperscript{st} NN = p_{14}

VR-1NN terminates but \textbf{BFS must examine} \(N_3\)

\(\Rightarrow D(q, p_{14}) = 5 > \text{mindist}(q, N_3)\)
Lemma: 2\textsuperscript{nd} NN of q is one of Voronoi neighbors of the 1\textsuperscript{st} NN of q.

candidate 2\textsuperscript{nd} NNs = \{p_4, p_8, p_{13}, p_{12}\}

Finding more NNs by navigating Voronoi diagram

\(\Rightarrow 1\textsuperscript{st} NN = p_{14}\)

\(\Rightarrow 2\textsuperscript{nd} NN = p_4\)
Lemma: kth NN of q is Voronoi neighbor of one of 1st, 2nd, ..., k-1th NN of q.

candidate 3rd NNs = \{p_8, p_{13}, p_{12}, p_5, p_6, p_7\}
VR-kNN

Performance Improvements:
- Using Voronoi cells for 1NN
  - e.g., no access to $N_3$
- Using Voronoi neighbors for kNN
  - e.g., no access to $N_2$ and $N_3$ for $k < 5$

I/O Complexity:
$O(\Phi(|P|) + k)$ where $\Phi(|P|)$ is the complexity of finding the 1st NN of $q$
kANN: k Aggregate Nearest Neighbor

- Given: $Q = \{q_1, ..., q_n\}$, integer $k$, and aggregate distance $f$
- $adist(p, Q) = f(D(p, q_1), ..., D(p, q_n))$
- Goal: find $k$ data points $p$ with smallest $adist(p, Q)$
- $f = \text{sum} \rightarrow$ the points that minimize the total distance to $Q$
- $f = \text{max} \rightarrow$ the points that minimize max distance to $Q$
- Variations: weighted sum, ...
kANN

- R-tree-based Algorithm: MBM [Papadias et al, TODS’05]
- Similar to BFS for kNN
- Heuristics to prune nodes
  - Lower bounds on \( \text{adist}(p', Q) \):
    - \( \text{adist}(p', Q) = f(D(p', q_1), ...) \geq \text{amindist}(N, \text{MBR}(Q)) = f(\text{mindist}(N, \text{MBR}(Q)), ...) \)
    - \( \text{adist}(p', Q) = f(D(p', q_1), ...) \geq \text{amindist}(N, Q) = f(\text{mindist}(N, q_1), ...) \)
- Problem: too conservative
  - No optimal coverage of SR

Assume: candidate result = \( p \)

\( \text{MBR}(Q) \)

\( q_1 \)

\( q_2 \)

\( q_3 \)

\( N \)

\( p' \)

\( > \text{adist}(p, Q) \rightarrow \text{do not access } N \)

\( > \text{adist}(p, Q) \rightarrow \text{do not access } N \)
VR-kANN

- Search Region of $p$ for $f=sum$
  $p'$ where $\text{adist}(p', Q) \leq \text{adist}(p, Q)$
- Co-circular areas for many functions
- VR-kANN’s two steps:
  1. Find a point close to the 1$^{\text{st}}$ ANN of $Q$
     
     $q \in R^2 = \text{centroid of } Q \text{ that minimizes } \text{adist}() = \text{center of all SRs}$
  2. Traverse the space using Voronoi diagram to finalize the result
- To ensure the coverage
  $\text{adist}(p', Q) \geq \text{amin\text{dist}}(V(p), Q)$

Use to check that $V(p)$ is intersecting SR
VR-kANN

F(p): lower bound of sum(p’,Q) for p’ in V(p)

• find b, the closest point to centroid q (use VR-1NN)

• add b’s neighbors into a minheap H ordered by F()

• add each visited point to candidate result

• iterate: remove the top, add its neighbors to H

• STOP condition: return a candidate a when adist(a,Q) <= key of top of H (we’ve covered b’s SR)

• NOTE: key of top of H is lower bound on sum() for all points extracted points.

\[ \min(\text{amindist}(V(p'),Q)) \Rightarrow \text{we have covered p’s SR} \]

Continue to cover more SRs and find more results
VR-kANN

Performance Improvements:

- Using Voronoi cells to cover SR
  - e.g., heuristics used by MBM
    [Papadias et al., TODS 30(2), 2005] suggests to examine N but no access to N in VR-kANN

I/O Complexity:

\[ O(\Phi(|P|) + k) \]

where \( \Phi(|P|) \) is the complexity of finding the cell including centroid \( q \)
RkNN: Reverse k Nearest Neighbor Query

- **Given:** point $q$ and int $k$
- **Goal:** find the data points that have $q$ as one of their $k$ NN; points $p$ in $P$ where $D(q,p) \leq D(q,p_k)$ where $p_k$ is $k$-th NN of $p$
- **R-tree-based Algorithm:** TPL [Tao et al., VLDB’04]
- **VR-RkNN:** Locate $q$ in $VD$, navigate to the points less than $k$ points away from $q$, stop when $q$’s $k$NN in each sector is found.
  - $L1$: $k$-th RNN of $q$ is in less than $k$ distance from $q$
  - $L2$ [Stanoi et al., VLDB’01]: RkNN of $q$ is one of $q$’s $k$NNs in each partition $S$
Spatial Skyline Query [VLDB’06, TODS’09]

- Given: \( Q = \{ q_1, ..., q_n \} \)
- Goal: find data points \( p \) for which there is no point closer than \( p \) to all \( q_i \)'s
- R-tree-based Algorithm: 
  \( B^2S^2 \) [Sharifzadeh et al., VLDB’06]
- Voronoi-based Algorithm: 
  \( VS^2 \) [Sharifzadeh et al., VLDB’06]
- VR-S\(^2\): similar to \( VS^2 \) and VR-kANN
- Improvement over \( B^2S^2 \) and \( VS^2 \)
  - I/O-optimality
  - Ability to report in the order of given function
Performance Evaluation

- **Real-world datasets (data points):**
  - **USGS** including one million locations in U.S.
  - **NE** including 124K locations in New York, Philadelphia and Boston

- **Methodology:** issuing 1000 NN queries of each type with random query points

- **Evaluating VoR-tree-based algorithms**
  - Number of accessed disk pages (I/O cost)

- **Parameters**
  - Size of result set (k) for kNN, RkNN, and kANN
  - Number of query points (|Q|) for kANN and SSQ
  - Extent of query points (size of MBR(Q)) for kANN and SSQ

- **Competitor approaches:**
  - BFS [Hjaltson et al., TODS 1999] for kNN
  - MBM [Papadias et al., TODS 30(2), 2005] for kANN
  - TPL [Tao et al., VLDB’04] for RkNN
Performance Evaluation

- Dataset: USGS
- I/O cost of VR-kNN
- Competitor approach:
  - BFS that utilizes an R-tree on data points

- VR-kNN examines less number of disk pages when k grows
- Up to 18% improvement for large k
Performance Evaluation

- Dataset: USGS
- I/O cost of VR-kANN
- Competitor approach:
  - MBM that utilizes an R-tree on data points

- Up to 64% improvement for VR-kANN
- VR-kANN’s I/O is almost half of MBM’s for small k
- for large k, they converge
Performance Evaluation

- Dataset: USGS
- I/O cost of VR-RkNN
- Competitor approach:
  - TPL that utilizes an R-tree on data points
    - Logarithmic scale
    - VR-RkNN’s I/O is much less than TPL (0.1% even for small k)
    - TPL uses a very conservative filter because the best theoretical filter is very complex to compute so it collects large candidate sets. VR-RkNN instead used Voronoi neighborhood information.
    - TPL examines almost all pages for large k
Summary and Future Directions

• We designed VoR-tree = R-tree + Voronoi diagram
• We developed I/O-efficient algorithms for NN queries
• We showed that our algorithms outperform their R-tree-based competitors
• Future Work:
  – Utilizing VoR-tree for other spatial spaces
  – Extending algorithms for non-point datasets
References