Voronoi-Based K Nearest Neighbor Search for Spatial Network Databases

Instructor: Cyrus Shahabi
Agenda

• Problem Definition
• Related Work
• Voronoi Diagrams
• Voronoi Based Network Nearest Neighbor (VN³)
• Experiments
• Discussion
Problem Definition

**kNN Problem:**
Given a set of objects $P$ and query point $q$, find the $k$ objects in $P$ that are closest to $q$. 
Related Work

- **Spatial Database**
- **kNN Query Processing**
- **Euclidean Space**
- **Road Networks**

- ✔ **NN Query**: Roussopoulos et al., SIMGOD 1995
- ✔ **K-NN for moving query point**: Zong et al., SSTD 2001
- ✔ **Time-parameterized queries**: Tao et al., SIMGOD 2002
- ✔ **Continuous NN Search**: Tao et al., VLDB 2002

Object Based
Related Work

- Sina-Continuous querying: Mokbel et al., SIGMOD 2004
- Sea-CNN queries: Xiong et al., ICDE 2005
- Monitoring kNN on moving objects: Yu et al., ICDE 2005
- Conceptual partitioning: Mouratidis et al., SIGMOD 2005

Space Based
Example 1:
Finding the 3 closest shopping centers
Example 2:
Finding the 3 closest restaurants to USC with Yahoo
Related Work

- Road Network Embedding for K-NN Search: Shahabi et al., ACM-GIS 2002
- Query processing in SNDB: Papadias et al., VLDB 2003
- Voronoi-based kNN in SNDB: Kolahdouzan et al., VLDB 2004
Related Work

Query processing in SNDB: Papadias et al., VLDB 2003
Incremental Network Expansion (INE)
- Blind expansion hence redundant node access
Related Work

Query processing in SNDB: Papadias et al., VLDB 2003
  – Incremental Network Expansion
Preliminaries: Voronoi Diagram

- Given a set of sites (POI), a Voronoi diagram partitions the plane into disjoint Voronoi polygons, one for each site.
- The region including a site $p$ includes all locations which are closer to $p$ than to any other object $p'$. 

$q$ is closer to the generator of the Voronoi Polygon than any other generator.
Preliminaries: Voronoi Properties

- **Property 1**: Voronoi diagram is unique
- **Property 2**: Voronoi edges are
  - shared by two generators
  - in equal distance to neighboring generators
- **Property 3**: Nearest generator of \( p \) is among the generators whose VP shares similar edges with \( p \)
- **Property 4**: Average number of Voronoi edges per VP is at most 6
Network Voronoi Diagram

Border Point: equal network distance to adjacent generators

Network Voronoi Polygon

Network Edge
VN³ Approach

• Offline Index Generation
  1. Network Voronoi Construction
  2. Index Generation (R-tree)
  3. Distance Precomputation

• Online Query Processing
  1. Find 1\textsuperscript{st} NN
  2. Find k NN-> Filter & Refine
Offline Step

• How to compute distance from q to candidates?
• Break distance from q to any P_i to:
  1. Distance from q to its borders
     I. Online Progressive Expansion
     II. Offline Pre-calculation
  2. Distance from borders of q to borders of P_i
     I. NVP Expansion
     II. Distance Computing Optimization
**Offline Step**

- Compute Network Voronoi Polygons (NVP)
- Index NVPs with R-Tree
- Precompute the intra (across polygons) and inter (within polygons) network distances for each NVP

1. Precompute the Shortest Path distance from each generator to its border point
2. Precompute the Shortest Path between border points

Example:

\[
d_n(q, P_9) = \min(d_n(q, P_9), d_n(q, P_9) + d_n(P_9, P_9), d_n(q, P_9) + d_n(q, P_9))
\]

\[
d_n(q, b_{34}) = \min(d_n(q, b_{34}), d_n(q, b_{34}) + d_n(b_{34}, P_9), d_n(q, b_{34}) + d_n(q, b_{34}))
\]

\[
d_n(b_{34}, P_9) = \min(d_n(b_{34}, P_9), d_n(b_{34}, P_9) + d_n(P_9, P_9), d_n(b_{34}, P_9) + d_n(q, P_9))
\]

\[
d_n(b_{34}, b_{36}) = \min(d_n(b_{34}, b_{36}), d_n(b_{34}, b_{36}) + d_n(b_{36}, P_9), d_n(b_{34}, b_{36}) + d_n(q, P_9))
\]

...
Online Step

- Find 1\textsuperscript{st} NN
  - Search R-Tree to find the NVP that overlaps q
  - Report the generator of the NVP as the 1\textsuperscript{st} NN;

Cost: $O(\log n)$
Online Step

- 1\textsuperscript{st} NN = P\textsubscript{1}
- 2\textsuperscript{nd} NN ∈ \{ Neighbors( P\textsubscript{1} ) \}  
  (e.g., P\textsubscript{3})
- 3\textsuperscript{rd} NN ∈ \{ Neighbors( P\textsubscript{1} ) \overset{\cup}{\cup} Neighbors( P\textsubscript{3} ) \}  
  (e.g., P\textsubscript{2})
- 4\textsuperscript{th} NN ∈ \{ Neighbors( P\textsubscript{1} ) \overset{\cup}{\cup} Neighbors( P\textsubscript{3} ) \overset{\cup}{\cup} Neighbors( P\textsubscript{2} ) \}  
- .....  

GOOD NEWS! In theory average # of neighbors = O(5k+1)=O(k)  
In Practice: << 5 × K
Performance of VN$^3$

• Data set:
  – Road network in Los Angeles (from NavTeq)
    • 110,000 streets, 79,800 intersections
    • Different points of interest:
      – restaurants, auto services, schools, parks, shopping centers, hospitals

• Measured:
  – Query response time and CPU
    • Comparison with INE [Papadias et al. (vldb03)]
  – Size of candidate set of VN$^3$ filter
    • Comparison with R-tree-based KNN [Seidl et al. (sigmod98) and Hjaltason et al. (tods99)]
## Pre-computation Overhead of VN³

<table>
<thead>
<tr>
<th>Entities</th>
<th>Cardinality</th>
<th>Points Inside each NVP</th>
<th>Average BPs per NVP</th>
<th>No. of Pre-comp. Border to Border</th>
<th>No. of Pre-comp. Border to inside</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hospitals</td>
<td>0.04%</td>
<td>1698</td>
<td>52</td>
<td>232,000</td>
<td>8,781,000</td>
</tr>
<tr>
<td>Shopping</td>
<td>0.16%</td>
<td>458</td>
<td>25</td>
<td>225,600</td>
<td>4,653,000</td>
</tr>
<tr>
<td>Parks</td>
<td>0.53%</td>
<td>142</td>
<td>14</td>
<td>239,500</td>
<td>2,630,000</td>
</tr>
<tr>
<td>Schools</td>
<td>1.15%</td>
<td>64</td>
<td>10</td>
<td>246,000</td>
<td>1,787,000</td>
</tr>
<tr>
<td>Auto. Svc.</td>
<td>3.26%</td>
<td>38</td>
<td>7</td>
<td>239,900</td>
<td>1,611,000</td>
</tr>
<tr>
<td>Restaurants</td>
<td>5.80%</td>
<td>27</td>
<td>6</td>
<td>243,600</td>
<td>1,348,000</td>
</tr>
</tbody>
</table>

The naïve approach: 3.2 billion precomputations times in worse cases when sparse.
Comparison of VN³ and INE

- K = 1

<table>
<thead>
<tr>
<th>NavTeq Entities</th>
<th>Density, Cardinality</th>
<th>VN³ (cpu), db sec</th>
<th>INE (cpu), disk sec</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hospitals</td>
<td>0.04% , 46</td>
<td>(0), 0.018</td>
<td>(0.30), 12.4</td>
</tr>
<tr>
<td>Shopping</td>
<td>0.16% , 173</td>
<td>(0), 0.020</td>
<td>(0.090), 3.6</td>
</tr>
<tr>
<td>Parks</td>
<td>0.53% , 561</td>
<td>(0), 0.021</td>
<td>(0.030), 1.4</td>
</tr>
<tr>
<td>Schools</td>
<td>1.15% , 1230</td>
<td>(0), 0.027</td>
<td>(0.015), 0.6</td>
</tr>
<tr>
<td>Auto. Svc.</td>
<td>3.26% , 2093</td>
<td>(0), 0.030</td>
<td>(0.013), 0.67</td>
</tr>
<tr>
<td>Restaurants</td>
<td>5.80% , 2944</td>
<td>(0), 0.032</td>
<td>(0.010), 0.49</td>
</tr>
</tbody>
</table>

Almost instantly increases for sparse
## Comparison of VN³ and INE

- **K = 5**

<table>
<thead>
<tr>
<th>Entities</th>
<th>Density, Cardinality</th>
<th>VN³ (cpu) , db sec</th>
<th>INE (cpu) , disk sec</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hospitals</td>
<td>0.04% , 46</td>
<td>(1.5) , 6.5</td>
<td>(1.7) , 78.3</td>
</tr>
<tr>
<td>Shopping</td>
<td>0.16% , 173</td>
<td>(0.45) , 3.3</td>
<td>(0.5) , 21.1</td>
</tr>
<tr>
<td>Parks</td>
<td>0.53% , 561</td>
<td>(0.15) , 1.5</td>
<td>(0.2) , 8.2</td>
</tr>
<tr>
<td>Schools</td>
<td>1.15% , 1230</td>
<td>(0.06) , 0.75</td>
<td>(0.07) , 3.5</td>
</tr>
<tr>
<td>Auto. Svc.</td>
<td>3.26% , 2093</td>
<td>(0.01) , 0.47</td>
<td>(0.05) , 2.43</td>
</tr>
<tr>
<td>Restaurants</td>
<td>5.80% , 2944</td>
<td>(0.01) , 0.87</td>
<td>(0.03) , 1.5</td>
</tr>
</tbody>
</table>

Database time is still dominant.

 sınav排除 K=1, VN³ is between 2 (high densities / larger K) to 14 (low densities / larger K) times faster than INE.
VN^3 vs INE

VN^3 finds the 1st NN using R-Tree
INE needs to expand the network around q

CPU Time:
INE uses a queue which is incrementally updated
VN³: Performance of Filter Step

- USGS Data, Euclidean space

- Ratio of size of candidate set to K decreases as K increases
  - Some of the new neighbors are already explored!

- VN³'s filter Behaves independently from density/distribution of points of interest
  - Size of the candidate set ( O(k) ) is independent of the density/distribution

- Better selectivity as compared to other approaches (up to 4 times)

- Very low variance (predictable behavior)
Conclusion

- A novel approach for KNN queries in SNDB
- Based on:
  - Pre-calculating the solution space (first order Voronoi diagrams)
  - Pre-computing some distances (from borders to points inside each polygon)
- Outperforms the only other solution for SNDB
- Independent from object distribution
- Outperforms the solutions for Euclidean space in “filtering”
Discussion

• What if the edge weights are changing?
• What if the both query and data objects are moving?
• What if you like to find the nearest hotel and gas station at the same time?
References


• A presentation by Ugur Demiryurek in csci587 Fall’2010