Edge-Based Locality Sensitive Hashing for Efficient Geo-Fencing Applications

Yi Yu†, Suhua Tang‡, Roger Zimmermann†

† Media Management Research Lab, School of Computing, National University of Singapore
‡ ATR Adaptive Communications and Research Laboratories, Kyoto

ACM SIGSPATIAL GIS 2013, Nov. 7, 2013
Orlando, Florida, USA
Outline

- Application and motivation of geo-fencing
  - Pairing a point with polygon: INSIDE/WITHIN
  - Crossing number algorithm and its scalability problem

- Proposed edge-based LSH algorithm
  - R-tree for pre-filtering
  - LSH for INSIDE, plus probing for WITHIN
  - Simple but effective and efficient

- Experimental results

- Conclusions
Motivation & Concepts: INSIDE

- **Basis:** Well-known crossing number algorithm
  - Inside iff number of intersections == odd
  - Requires checking each edge ⇒ inefficient

- **Enhancements:**
  1) Exploiting MBR for pre-filtering
  2) Locality-sensitive hashing (LSH) for further acceleration

**MBR:** minimum bounding rectangle
Motivation & Concepts: WITHIN

- Case: Point $P$ outside of MBR but within a distance of $d_{th}$
  - A rectangle centered at $P$, edge length being two times $d_{th}$
  - If no overlap $\Rightarrow$ point surely not WITHIN distance
Scalability: Polygons, Points, Edges

# edges of 15 polygons

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>285</td>
<td>255</td>
<td>235</td>
<td>196</td>
<td>264</td>
<td>250</td>
<td>240</td>
<td>239</td>
<td>226</td>
<td>226</td>
<td>242</td>
<td>153, 15</td>
<td>152, 20</td>
<td>250</td>
<td>217</td>
</tr>
</tbody>
</table>

⇒ Problem: scalability with the number of edges
Efficient Geo-Fencing: Framework

- Two stages
  1) R-tree-based pre-filtering
  2) LSH adapted to crossing number algorithm

Diagram:

- Polygon file
  - Polygon management
    - MBRs in R-tree
      - Polygon cache
      - Edges in LSH table
  - R-tree based pre-filtering
    - Pairing engine
      - LSH-based INSIDE detection
        - INSIDE result
      - LSH-based WITHIN detection
        - WITHIN result
Efficient Geo-Fencing: R-tree Based Pre-filtering
Efficient Geo-Fencing: INSIDE Detection

- A separate hash table for each polygon
- A fixed number of buckets, N, for each hash table
- Hash function
  - \( T = \frac{X_{\text{max}} - X_{\text{min}}}{N} \)
  - \( \text{HashKey}(x) = \text{int} \left( \frac{(x - X_{\text{min}})}{T} \right) \)
- An edge \((x_1, y_1)\)—\((x_2, y_2)\) stored in buckets from key1 to key2,
  - key1=HashKey(x1), key2=HashKey(x2)
Efficient Geo-Fencing: WITHIN Detection

- **LSH with multi-probing**
  - Inside polygon $\bullet P_1$
  - Inside inner ring $\bullet P_2$
  - Outside outer ring $\bullet P_3$

- **Optimization $P_3$**
  - Range of a point
  - Divide outer area into 4 ranges
  - Only check edge in the same range with the point

Diagram:
- Buckets probed for WITHIN $d_{th}$ of polygon
- Points $P_L$, $P_T$, $P_B$, $P_R$
Geo-Fencing: Evaluation Setup

- **Training dataset**
  - Two point files: Point500 (39,289 instances), Point1000 (69,619 instances)
  - Two polygon files: Poly10 (30 instances), Poly15 (40 instances)
  - Ground truth available (different combinations of inputs and predicates)

- **Two predicates**
  - INSIDE & WITHIN 1000
  - Execution times without overhead (file I/O, data conversion)
  - Accuracy & efficiency (4 methods)

- **Environment**
  - A laptop PC (Intel Core i5 CPU, 64-bit Windows 7)
Geo-Fencing: Example Experiments

- 100% accuracy with test set
- Running time without system overhead
  - Measured via Windows `QueryPerformanceCounter()`: 100 runs
  - LSH+R-tree: Execution speed-up by 970% for INSIDE and by 370% for WITHIN

![Bar charts showing average time (ms) for different predicates and methods](image-url)
Other Optimizations

- Execution profiling showed that I/O processing required considerable time
  - Large amounts of text data needed to be read

- Therefore we applied several I/O optimizations
  - Reading data in larger blocks, not line-by-line
  - Writing pairing results in large blocks
  - Optimized number conversion: text-float to binary-float
  - Multi-threading

- Batch processing
  - Multiple points at a time, find candidate pairs for each polygon
  - Precise pairing for each polygon (CPU cache optimization)
Conclusions

- Different levels of approximation
  - Polygon as MBR: pre-filtering via R-tree
  - Edges in bucket: LSH

- LSH table per polygon
  - Compatible with R-tree
  - Fixed number of buckets, less affected by the distribution and shapes of polygons

- Simple, effective and efficient
  - 100% accuracy, high speed

Acknowledgments
The work presented was in part supported by the Singapore National Research Foundation under its International Research Centre @ Singapore Funding Initiative and administered by the IDM Programme Office.
Thank You – Q&A

Further information at:

http://eiger.ddns.comp.nus.edu.sg

yuy@comp.nus.edu.sg
rogerz@comp.nus.edu.sg