Spatial Keyword Range Search on Trajectories

Yuxing Han¹, Liping Wang¹, Ying Zhang², Wenjie Zhang³, Xuemin Lin¹,³

¹East China Normal University, China
²University of Technology, Sydney
³The University of New South Wales, Sydney
Big Trajectory Data

- Wireless Sensors with Global Position System
- Driving Route Track Record
- Social Network with Location-based Service
Motivation

It is meaningful to search trajectories based on three aspects, i.e., spatio, temporal, textual.

- **query keywords**
- **query range**
- **query timespan**

(last month)
Related Work

✦ Activity Trajectory Similarity Query (ATSQ)


✦ Top-\(k\) Spatial Keyword Query (TkSK) on trajectory

Problem Statement

Spatial Keyword Range search on Trajectories (SKRT):

Given a query region, a timespan and a set of keywords, we aim to retrieve trajectories that go through this region during query timespan, and contain all the query keywords.
Example

\[ Q.R \] : the space within dotted circle

\[ Q.T = [t_s, t_e] \]

\[ Q.\Phi = \{a, c\} \]
Inverted Octree (IOC-Tree)

- inverted index
- octree
- morton code
- signature technique
Inverted Index

**WHY** follow keyword-first-pruning?
Octree & Morton Code

✦ octree: 3D analog of quadtree

✦ morton code: two nodes with high spatio-temporal proximity are assigned to the same page in the secondary storage
Signature

- for each node in octree, a *signature* is maintained to **summarize** the identifications of the trajectories that go through the corresponding spatio-temporal region.
IOC-Tree of Example

Table 2. Distribution of Trajectory Points

<table>
<thead>
<tr>
<th>Node#</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Points</td>
<td>$p_{11}$</td>
<td>$p_{10}$</td>
<td>$p_{30}$</td>
<td>$p_{20}$</td>
<td>$p_{23}$</td>
<td>$p_{12}$</td>
<td>$p_{22}$</td>
<td>$p_{42}$</td>
</tr>
<tr>
<td></td>
<td>$p_{31}$</td>
<td>$p_{41}$</td>
<td>$p_{24}$</td>
<td>$p_{13}$</td>
<td>$p_{32}$</td>
<td>$p_{33}$</td>
<td>$p_{14}$</td>
<td>$p_{34}$</td>
</tr>
</tbody>
</table>

$OC_a$

$OC_c$
Algorithm

Main Idea: Prune as many trajectories as possible based on spatio, temporal, textual info by using IOC-Tree

✦ Prune & Verification Strategy

✦ Three types of octree nodes:
  a) ones locate outside the range,
  b) fully-covered nodes,
  c) partially-covered nodes.
Pruning

✧ Procedure Prune prunes non-promising nodes based on IOC-Tree

✧ **STRangeFilter**: only explore black nodes (i.e., non-empty nodes)

✧ deals with nodes level by level among related octrees

---

**Procedure Prune(Q, L)**

1. while \( L \neq \emptyset \) do
2.    **STRangeFilter**(Q.R, Q.T, L);
3.    foreach \( k_i \in Q.\delta \) do
4.       \( SIG_i \) = bitwise-OR of signatures of nodes \( v \in L \) from \( OC_i \);
5.    foreach node \( v \in L \) from \( OC_i \) do
6.       foreach \( SIG_j \) where \( j \neq i \) do
7.          SignatureCheck(v, SIG_j);
8.    foreach node \( v \in L \) that survive from the signature test do
9.       Suppose \( v \) comes from \( OC_j \);
10.      if \( v \) is a fully covered leaf node then
11.         add \( v \) into \( CN^f_j \);
12.      else if \( v \) is a partially covered leaf node then
13.         add \( v \) into \( CN^p_j \);
14.      else if \( v \) is non-leaf node then
15.         foreach child node \( v' \) of \( v \) do
16.            if \( v' \) is not a white node then
17.               put \( v' \) into \( L \);
18.         delete \( v \) from \( L \);
19.    foreach \( k_i \in Q.\delta \) do
20.      determine \( TR(CN^f_i) \) and \( TR(CN^p_i) \);
Procedure **Verification** aims at validating candidate trajectories.

Further validation should load corresponding cell on the disk.
Extension

- Spatial Keyword Range search on Trajectories with Order-sensitive keywords (SKRTO)
Experiment

✦ Implemented in C++

✦ Windows 7
   · Intel i5 CPU(3.10GHz)
   · 8 GB main memory

✦ Three Real Trajectory Datasets:
   a) two Foursquare check-in datasets from LA and NY
   b) one geo-tagged tweets dataset

✦ Two Baselines:
   a) GAT (ICDE’13)       b) B^{ck}-tree (arxiv’12)
## Dataset Statistics

**Table 3. Dataset Statistics**

<table>
<thead>
<tr>
<th></th>
<th>LA</th>
<th>NY</th>
<th>TW</th>
</tr>
</thead>
<tbody>
<tr>
<td>#trajectory</td>
<td>31,553</td>
<td>49,022</td>
<td>214,834</td>
</tr>
<tr>
<td>#location</td>
<td>215,614</td>
<td>206,416</td>
<td>1,287,315</td>
</tr>
<tr>
<td>#tag</td>
<td>3,175,597</td>
<td>3,068,401</td>
<td>28,645,905</td>
</tr>
<tr>
<td>#distinct-tag</td>
<td>100,843</td>
<td>89,665</td>
<td>1,852,141</td>
</tr>
</tbody>
</table>

## Experimental Settings

**Table 4. Experimental Settings**

|       | $|Q.\phi|$ | $|Q.T|$ (month) | $\delta(Q.R)$ (km) |
|-------|------------|----------------|---------------------|
| LA, NYC| 2, 3, 4, 5  | 1, 2, 3, 4, 5  | 10, 20, 30, 40, 50   |
| TW    | 2, 3, 4, 5  | 0.5, 1, 1.5, 2, 2.5| 5, 10, 15, 20, 25   |
Varying number of query keywords

(a) $SKRT$ on LA
(b) $SKRT$ on NYC
(c) $SKRT$ on TW
Varying query timespan

(a) SKRT on LA
(b) SKRT on NYC
(c) SKRT on TW

(a) SKRTO on LA
(b) SKRTO on NYC
(c) SKRTO on TW
Varying query range

(a) $SKRT$ on LA
(b) $SKRT$ on NYC
(c) $SKRT$ on TW

(a) $SKRTO$ on LA
(b) $SKRTO$ on NYC
(c) $SKRTO$ on TW
Conclusion

✧ Propose spatial keyword range search on trajectories

✧ Design IOC-Tree structure and query processing algorithm

✧ Extensive experiments on real datasets confirm the efficiency of our techniques
Thank you!