Case Study: Trails

Binxuan Huang
binxuanh@cs.cmu.edu

CASOS
Societal Computing, School of Computer Science, Carnegie Mellon University

- Introduction of trail, Markov transition network
- Case study of health trails
- Current research progress
- Hands on session
**Introduction**

- **Trail**: a path of an object through time and space

<table>
<thead>
<tr>
<th>Time</th>
<th>4 pm@Apr. 1</th>
<th>3 pm@Apr. 2</th>
<th>9 am@Apr. 3</th>
<th>1 pm@Apr. 3</th>
<th>2 pm@Apr. 4</th>
<th>4 pm@Apr. 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trail 1</td>
<td>L1</td>
<td>L2</td>
<td>L3</td>
<td>L2</td>
<td>L1</td>
<td>L2</td>
</tr>
<tr>
<td>Trail 2</td>
<td>L2</td>
<td>L3</td>
<td>L4</td>
<td>L2</td>
<td>L1</td>
<td>L1</td>
</tr>
<tr>
<td>Trail 3</td>
<td>L2</td>
<td>L3</td>
<td>L1</td>
<td>L1</td>
<td>L2</td>
<td>L3</td>
</tr>
</tbody>
</table>

![Traffic flow network](image1)

- Adding **BEGIN/END** locations to capture more information

<table>
<thead>
<tr>
<th>Time</th>
<th>4 pm@Apr. 1</th>
<th>4 pm@Apr. 2</th>
<th>3 pm@Apr. 2</th>
<th>9 am@Apr. 3</th>
<th>1 pm@Apr. 3</th>
<th>2 pm@Apr. 4</th>
<th>4 pm@Apr. 5</th>
<th>4 pm@Apr. 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trail 1</td>
<td>BEGIN</td>
<td>L1</td>
<td>L2</td>
<td>L3</td>
<td>L2</td>
<td>L1</td>
<td>L2</td>
<td>END</td>
</tr>
<tr>
<td>Trail 2</td>
<td>BEGIN</td>
<td>L2</td>
<td>L3</td>
<td>L4</td>
<td>L2</td>
<td>L1</td>
<td>L1</td>
<td>END</td>
</tr>
<tr>
<td>Trail 3</td>
<td>BEGIN</td>
<td>L1</td>
<td>L2</td>
<td>L3</td>
<td>L1</td>
<td>L2</td>
<td>L3</td>
<td>END</td>
</tr>
</tbody>
</table>

![Traffic flow network](image2)
Case study

- Case study: one patient health trail
- Case study: one patient health trail with BEGIN/END nodes
Case study

- Compare between one patient health trail with/without BEGIN/END
- Without BEGIN/END
- With BEGIN/END

Case study

- Compare between group of health trails with/without BEGIN/END
- Without BEGIN/END
- With BEGIN/END
### Current progress: Overcome the Low Time Resolution in Trails

**Time** | 4 pm @ Apr. 1 | 3 pm @ Apr. 2 | 9 am @ Apr. 3 | 1 pm @ Apr. 3 | 2 pm @ Apr. 4 | 4 pm @ Apr. 5  
--- | --- | --- | --- | --- | --- | ---  
Trail 1 | L1 | L2 | L3 | L2 | L1 | L2  
Trail 2 | L2 | L3 | L4 | L2 | L1 | L1  
Trail 3 | L2 | L3 | L3 | L1 | L1 | L2  

**Lower time resolution**  
**Broken point**  

### Current progress: Overcome the Time Resolution Issue in Trails

- For a location sequence in a broken point: Find a location sequence with maximum transition probability product  
  \[ p(l_1 \rightarrow l_2 \rightarrow \cdots \rightarrow l_n) = p(l_2 | l_1)p(l_3 | l_2) \cdots p(l_n | l_{n-1}) \]  
- Relation with Asymmetric Traveling Salesman Problem (ATSP): visit each location exactly once and find the minimum travelling cost (NP-hard)
Current progress: Overcome the Time Resolution Issue in Trails

- Four algorithms
  - Random: Randomly pick the location order
  - Greedy: At each step, select next location with maximum transition probability.
  - Exact algorithm: enumerate all the possible routes.
  - Ant Colony System (ACS): an approximate algorithm designed for ATSP

- Partition trails when time intervals are larger than a threshold.

<table>
<thead>
<tr>
<th>Time</th>
<th>Apr. 1</th>
<th>Apr. 1</th>
<th>Apr. 2</th>
<th>Apr. 3</th>
<th>June 3</th>
<th>June 4</th>
<th>June 5</th>
<th>June 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trail 1</td>
<td>BEGIN</td>
<td>L1</td>
<td>L2</td>
<td>L3</td>
<td>L2</td>
<td>L1</td>
<td>L2</td>
<td>END</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time</th>
<th>Apr. 1</th>
<th>Apr. 1</th>
<th>Apr. 2</th>
<th>Apr. 3</th>
<th>Apr. 3</th>
<th>June 3</th>
<th>June 3</th>
<th>June 4</th>
<th>June 5</th>
<th>June 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trail 1</td>
<td>BEGIN</td>
<td>L1</td>
<td>L2</td>
<td>L3</td>
<td>END</td>
<td>BEGIN</td>
<td>L2</td>
<td>L1</td>
<td>L2</td>
<td>END</td>
</tr>
</tbody>
</table>

Current progress: Overcome the Time Resolution Issue in Trails

- Datasets:
  - Health record data
    - Location: health service
    - Agent: patient
    - Record: (patient, health service, date)

<table>
<thead>
<tr>
<th></th>
<th># of Records</th>
<th># of Agents</th>
<th># of Locations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Health data</td>
<td>94885</td>
<td>5055</td>
<td>115</td>
</tr>
</tbody>
</table>

Use 814 unbroken trails as testing data
Current progress: Overcome the Time Resolution Issue in Trails

• Statistics of health dataset

<table>
<thead>
<tr>
<th>Time resolution (day)</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>
</tr>
</thead>
<tbody>
<tr>
<td># of broken trails</td>
<td>164</td>
<td>213</td>
<td>250</td>
<td>263</td>
<td>268</td>
<td>278</td>
<td>294</td>
<td>297</td>
</tr>
<tr>
<td># of broken points</td>
<td>178</td>
<td>251</td>
<td>296</td>
<td>319</td>
<td>340</td>
<td>353</td>
<td>374</td>
<td>389</td>
</tr>
<tr>
<td>Avg. length of broken points</td>
<td>2.073</td>
<td>2.223</td>
<td>2.284</td>
<td>2.426</td>
<td>2.482</td>
<td>2.586</td>
<td>2.591</td>
<td>2.627</td>
</tr>
<tr>
<td>Avg. length of cont. broken points</td>
<td>2.121</td>
<td>2.364</td>
<td>2.449</td>
<td>2.650</td>
<td>2.749</td>
<td>2.916</td>
<td>2.909</td>
<td>3.041</td>
</tr>
<tr>
<td>Max. length of broken points</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>

Experiments setup
- Artificially change the time resolution of 814 unbroken health trails
  - $timestamp' = \left\lceil \frac{timestamp}{resolution} \right\rceil \ast resolution$. 
Current progress: Overcome the Time Resolution Issue in Trails

- The effects of BEGIN/END nodes and partitions

Comparison between four algorithms
Hands on

• Data preparation: program function call data

<table>
<thead>
<tr>
<th>id</th>
<th>timestamp</th>
<th>location</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>553987065457672.00</td>
<td>class.edu.umd.cs.findbugs.PluginLoader</td>
</tr>
<tr>
<td>0</td>
<td>553987065574331.00</td>
<td>class.edu.umd.cs.findbugs.PluginLoader</td>
</tr>
<tr>
<td>0</td>
<td>553987065768508.00</td>
<td>class.java.lang.Class</td>
</tr>
<tr>
<td>0</td>
<td>553987065819048.00</td>
<td>class.edu.umd.cs.findbugs.PluginLoader</td>
</tr>
<tr>
<td>0</td>
<td>553987100679470.00</td>
<td>class.java.net.URL</td>
</tr>
<tr>
<td>0</td>
<td>553987102005655.00</td>
<td>class.edu.umd.cs.findbugs.PluginLoader</td>
</tr>
<tr>
<td>0</td>
<td>553987102202112.00</td>
<td>class.edu.umd.cs.findbugs.PluginLoader</td>
</tr>
<tr>
<td>0</td>
<td>553987102260252.00</td>
<td>class.edu.umd.cs.findbugs.PluginLoader</td>
</tr>
<tr>
<td>0</td>
<td>553987102331691.00</td>
<td>class.edu.umd.cs.findbugs.PluginLoader</td>
</tr>
<tr>
<td>0</td>
<td>553987102384890.00</td>
<td>class.edu.umd.cs.findbugs.PluginLoader</td>
</tr>
<tr>
<td>0</td>
<td>553987102425550.00</td>
<td>class.edu.umd.cs.findbugs.PluginLoader</td>
</tr>
<tr>
<td>0</td>
<td>55398747619118.00</td>
<td>class.edu.umd.cs.findbugs.PluginLoader</td>
</tr>
<tr>
<td>0</td>
<td>553987476619437.00</td>
<td>class.java.net.URL</td>
</tr>
<tr>
<td>0</td>
<td>553987476664276.00</td>
<td>class.edu.umd.cs.findbugs.PluginLoader</td>
</tr>
<tr>
<td>0</td>
<td>553987476741035.00</td>
<td>class.java.lang.String</td>
</tr>
<tr>
<td>0</td>
<td>553987476797655.00</td>
<td>class.edu.umd.cs.findbugs.PluginLoader</td>
</tr>
</tbody>
</table>

• Data import

- Ego network: transition tables
• Data import

Step 1: Select an ego-network file:
The file must have an Ego node column, and one or more Path node columns. Each path column will produce a Path x Path transition network where link values record the number of times an ego transitioned from one path node to another.

/Users/jimmy/Desktop/SLI/program trail.csv

- Use only lines where column id has value:

Step 2: Select how the file is ordered:
- Rows are sorted first by ego name and then by date
- Sort rows by column timestamp which has Dates are the number of seconds since 19
- Ego is at multiple path nodes at once: Create all possible path links each of value one

Date processing options:
- Aggregate dates by 6 Hour(s)
- Window for transitions 6 Hour(s)

• Data import

Step 1: Select an ego column and optional entry/exit state columns:
- Ego column: id
- Class: Agent
- Name: Agent
- Create new ego nodes during import

Step 2: Select one or more path columns:
- id column:
- timestamp column:
- location column:
- Create new path nodes during import

Class: Location
Name: Location
Transition network name: Location x Location
Ego path network name: Agent x Location
Hands on

• Data transform

Flow network

Hands on

• Data transform
Hands on

• Data transform

- Symmetrize by method
- Binarize link values ($x \neq 0 \Rightarrow x = 1$)
- Collapse link values ($a \geq x \leq b \Rightarrow x = 1$)
- Negate link values ($-x$)
- Invert the link values ($1/x$)
- Logarithm of the link values ($\log_{10}(x)$)
- Absolute value of the link values ($|x|$)
- Scale the link values ($c \times x$)
- Row Sum Normalize the link values ($x_{i,j}/(\text{sum of row } i)$)
- Column Sum Normalize the link values ($x_{i,j}/(\text{sum of column } j)$)
- Sum Normalize the link values ($x_{i,j}/(\text{sum of all values})$)
- Increment the link values ($c + x$)
- Subtract link values ($c - x$)
- Remove self-loops (diagonal)

Hands on

• Analysis

Keyframe: Result Transitions

- Meta-Network Name: Result Transitions
- Meta-Network Time:
- Filename:
  - Generate Reports
  - Visualize
  - Measure Charts

General statistics: