Trajectory Network Mining

Manos Papagelis
papaggel@eeecs.yorku.ca
Current research focus

A. Trajectory Network Mining

B. Network Representation Learning

C. Streaming & Dynamic Graphs

D. Social Media Mining & Analysis

E. City Science / Urban Informatics / IoT

F. Natural Language Processing
Node Importance in Trajectory Networks
every moving object, forms a trajectory – in 2D it is a sequence of \((x, y, t)\) there are trajectories of moving cars, people, birds, ...
Trajectory data mining

trajectory similarity

trajectory clustering

trajectory anomaly detection
trajectory pattern pattern mining
trajectory classification
...more

we care about network analysis of moving objects
Proximity networks
Distance can represent

- line of sight
- wifi/bluetooth signal range
Trajectory networks
The problem

Input: logs of trajectories (x, y, t)
Output: node importance metrics
Node Importance
Node importance in static networks

- **Degree centrality**
- **Betweenness centrality**
- **Closeness centrality**
- **Eigenvector centrality**
Node importance in TNs

- Node degree *over time*
- Triangles *over time*
- Connected components *over time* (connectedness)
Applications

- infection spreading
- wireless signal security
- rich dynamic network analytics
Evaluation of Node Importance in Trajectory Networks
Naive approach
Naive approach

For *every* discrete time unit:

1. get static **snapshot** of network
2. run **static** node importance **algorithms** on snapshot

**Aggregate** results at the end
Streaming approach

Similar to naive, but:

- no final aggregation
- results calculated incrementally at every step

Still every time unit
Every discrete time unit
Sweep Line Over Trajectories (SLOT)
Sweep line algorithm

A computational geometry algorithm that given line segments computes line segment overlaps

Efficient one pass algorithm that only processes line segments at the beginning and ending points
**SLOT**: Sweep Line Over Trajectories

(algorithm sketch)

represent TN edges as time intervals

apply variation of sweep line algorithm

simultaneously compute node degree, triangle membership, connected components in one pass
Represent edges as time intervals

e_1: (n_1, n_2)

e_n

t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8, t_9, t_10, t_11, t_12, t_13, T
SLOT: Sweep Line Over Trajectories
At every edge start

- **node degree**
  - nodes $u$, $v$ now connected
  - increment $u$, $v$ node degrees

- **triangle membership**
  - did a triangle just form?
  - look for $u$, $v$ common neighbors
  - increment triangle $(u, v, \text{common})$

- **connected components**
  - did two previously unconnected components connect?
  - compare old components of $u$, $v$
  - if no overlap, merge them
At every edge stop

- **node degree**
  - nodes $u, v$ now disconnected
  - decrement $u, v$ degree

- **triangle membership**
  - did a triangle just break?
  - look for $u, v$ common neighbors
  - decrement triangle $(u, v, \text{common})$

- **connected components**
  - did a component separate?
  - BFS to see if $u, v$ still connected
  - if not, split component to two
**SLOT**: At the end of the algorithm ...

**node degrees**: start/end time, duration

**triangles**: start/end time, duration

**connected components**: start/end time, duration

**Exact** results (not approximations)

e.g. node degree of $u$ $d(u)$ is:

- $d(u) = 5$, from $t=0$ until $t=10$ duration $=10$
- $d(u) = 6$, from $t=10$ until $t=50$ duration $=40$
- $d(u) = 4$, from $t=50$ until $t=100$ duration $=50$
Evaluation of SLOT
Simulating trajectories

constant velocity

random velocity
Node degree

![Graph showing the comparison of Running Time (s) with respect to the number of objects for different methods: Naive, Streaming, and SLOT. The graph indicates a significant improvement in performance with SLOT, with a 1550x increase in processing efficiency compared to Naive.](image)
Triangle membership / connected components

![Graph showing running time vs number of objects]

- Naive-Connectedness
- Naive-Triangles
- SLOT-Connectedness
- SLOT-Triangles

The graph illustrates the running time (s) in relation to the number of objects. As the number of objects increases, the running time for both naive and SLOT methods increases, with SLOT showing a lower running time compared to naive methods.
SLOT Scalability

![Graph showing running time vs. number of objects for different metrics: Degree, Connectedness, and Triangle.](image)
Takeaway

trajectory networks

network importance over time

SLOT properties:
- fast
- exact
- scalable

SLOT algorithm
Seagull migration trajectories

data from Wikelski et al. 2015
Group Pattern Discovery of Pedestrian Trajectories
Pedestrian trajectories
what is a group?
many definitions, many algorithms

e.g., *flock, convoy, evolving-clusters, gathering-pattern*, … [ACM TIST Tutorial 2015]
Finding pedestrian groups

Local Grouping
Intuitive method
Spatial-only

key idea
find pairs of pedestrians \( x, y \) where \( \text{distance}(x, y) < \theta \)
expand pairs to discover groups
Local grouping
Challenge: Projection into ground plane

High perspective distortion - pedestrians closer to the camera appear larger than the ones farther away.

Estimated Homography to overcome this distortion.
expand the key idea to include the time dimension
Global groups vs. Time-window groups

global grouping
time-window grouping
Trajectolizer

Demo
Trajectolizer: System Overview

- Pedestrian Monitoring System
- Video Streams
- Pedestrian Annotation
- Raw (Pedestrian) Trajectory Streams

- Trajectories Visualization
- Trajectory Groups
- Trajectory Pattern Mining
- Refined Trajectories
Trajectolizer: Interactive Demo

- **Current frame with pedestrian IDs and trajectories**
- **Timeline slider area to navigate video frames**
- **Descriptive statistics about the current frame**
- **Grouping analysis**
EvoNRL: Evolving Network Representation Learning Based on Random Walks
networks

(universal language for describing complex data)
Classical ML Tasks in Networks

- Community Detection
- Link Prediction
- Node Classification
- Triangle Count
- Graph Similarity
- Anomaly Detection
Limitations of Classical ML Tasks

expensive computation
(high dimension computations)

extensive domain knowledge
(task specific)
Network Representation Learning (NRL)

- faster computations
  (low dimension computations)

- agnostic domain knowledge
  (task independent)
Network Representation Learning (NRL)

several network structural properties can be learned/embedded
(nodes, edges, subgraphs, graphs, ...)

Network

Low-dimension space
Random Walk-based NRL

Input network

Obtain a set of random walks

Learn a vector representation for each node

Feed sentences to Skip-gram NN model

Treat the set of random walks as sentences
Random Walk-based NRL

StaticNRL

DeepWalk
node2vec
...

54
But…

real-world networks are constantly changing
how can we learn representations of an evolving network?
Naive Approach

\[ t = 0 \]

\[ t = 1 \]

\[ t = 2 \]
Limitation #1

time expensive
Limitation #2

Random Walks

Neural Network Optimization

incomparable representations
EvoNRL Key Idea

Input network

Obtain a set of random walks

Learn a vector representation for each node

Feed sentences to Skip-gram NN model

Learn a set of random walks

Treat the set of random walks as sentences
dynamically maintain a set of random walks for every change in the network
Example

$t = 0$

![Graph at t = 0](image)

addition of edge (1, 4)

$t = 1$

![Graph at t = 1](image)

need to update the RW set

<table>
<thead>
<tr>
<th>1</th>
<th>3 5 8 7 6 4 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1 3 5 8 7 6 5</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>87</td>
<td>8 5 4 3 5 6 7</td>
</tr>
<tr>
<td>88</td>
<td>4 5 6 7 8 9</td>
</tr>
<tr>
<td>89</td>
<td>2 1 3 5 6 7 8</td>
</tr>
<tr>
<td>90</td>
<td>7 4 2 1 3 5 6</td>
</tr>
</tbody>
</table>

need to update the RW set

<table>
<thead>
<tr>
<th>1</th>
<th>3 5 8 7 6 4 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1 3 5 8 7 6 5</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>87</td>
<td>8 5 4 3 5 6 7</td>
</tr>
<tr>
<td>88</td>
<td>4 5 6 7 8 9</td>
</tr>
<tr>
<td>89</td>
<td>2 1 3 5 6 7 8</td>
</tr>
<tr>
<td>90</td>
<td>7 4 2 1 3 5 6</td>
</tr>
</tbody>
</table>
how can we efficiently maintain a set of random walks?
EvoNRL Operations

Operations on RW
- **Search** a node
- **Delete** a RW
- **Insert** a new RW

+ edge(n₁, n₂)
EvoNRL Indexing

Each node is a keyword
Each RW is a document
A set of RWs is a collection of documents

<table>
<thead>
<tr>
<th>Term</th>
<th>Frequency</th>
<th>Postings and Positions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>&lt;2, 1&gt;, &lt;89, 2&gt;, &lt;90, 4&gt;</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>&lt;89, 1&gt;, &lt;90, 3&gt;</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>&lt;1, 1&gt;, &lt;2, 1&gt;, &lt;87, 3&gt;, &lt;89, 3&gt;, &lt;90, 5&gt;</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>&lt;1, 6&gt;, &lt;87, 3&gt;, &lt;90, 2&gt;</td>
</tr>
<tr>
<td>5</td>
<td>9</td>
<td>&lt;1, 2&gt;, &lt;1, 7&gt;, &lt;2, 3&gt;, &lt;2, 7&gt;, &lt;87, 5&gt;, &lt;88, 2&gt;, &lt;89, 4&gt;, &lt;90, 6&gt;</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>&lt;1, 5&gt;, &lt;2, 6&gt;, &lt;87, 6&gt;, &lt;88, 3&gt;, &lt;89, 3&gt;, &lt;90, 5&gt;</td>
</tr>
<tr>
<td>7</td>
<td>5</td>
<td>&lt;1, 4&gt;, &lt;2, 5&gt;, &lt;87, 7&gt;, &lt;88, 4&gt;, &lt;89, 6&gt;, &lt;90, 7&gt;</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
<td>&lt;1, 3&gt;, &lt;2, 4&gt;, &lt;87, 1&gt;, &lt;88, 6&gt;, &lt;89, 7&gt;</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>&lt;88, 7&gt;</td>
</tr>
</tbody>
</table>
Evaluation: EvoNRL vs StaticNRL

Accuracy
- EvoNRL ≈ StaticNRL

Running Time
- EvoNRL << StaticNRL
EvoNRL has the **similar accuracy** as StaticNRL.
EvoNRL performs orders of time faster than StaticNRL
Summary

how can we learn representations of an evolving network?

EvoNRL

time efficient
accurate
generic method
Thank you!
Questions?
Credits

Farzaneh Heidari

Tilemachos Pechlivanoglou

Abdullah Sawas

Mahmoud Afifi

Abdullah Abuolaim

Data Mining Lab @ YorkU
References


