Trajectory Based Traffic Analysis (Demo Paper)

Benjamin Krogh
Dept. of Computer Science
Aalborg University, Denmark
bkrogh@cs.aau.dk

Ove Andersen
Dept. of Computer Science
Aalborg University, Denmark
xcalibur@cs.aau.dk

Edwin Lewis-Kelham
Dept. of Computer Science
Aalborg University, Denmark
edwin@cs.aau.dk

Nikos Pelekis
Dept. of Statistics and
Insurance Science
University of Piraeus, Greece
npelekis@unipi.gr

Yannis Theodoridis
Dept. of Informatics
University of Piraeus, Greece
ytheod@unipi.gr

Kristian Torp
Dept. of Computer Science
Aalborg University, Denmark
torp@cs.aau.dk

ABSTRACT

We present the INTRA system for interactive path-based traffic analysis. The analyses are developed in collaboration with traffic researchers and provide novel insights into conditions such as congestion, travel-time, choice of route, and traffic-flow. INTRA supports interactive point-and-click analysis, due to a novel and efficient indexing structure. With the web-site `daisy.aau.dk/its/spqdemo/` we will demonstrate several analyses, using a very large real-world data set consisting of 1.9 billion GPS records (1.5 million trajectories) recorded from more than 13 000 vehicles, and touching most of the road network in Denmark.

Categories and Subject Descriptors

H.2.8 [**Database Applications**]: Spatial databases and GIS; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

General Terms

Design, Algorithms, Experimentation, Measurement

1. INTRODUCTION

The trajectories of moving objects in a road network are a good data source for traffic analysis, because the *paths* can be included into the analysis, in contrast to point-based measurements. For instance, with a large representative set of trajectories, it is possible to compute the average turn-time of intersections, travel times along paths, paths traveled between an origin and destination, and a wide range of more specialized analyses such as stop-behavior for a sequence of signalized intersections. Some of these analyses require retrieving the trajectories that strictly follow a specific path, i.e., without detours, while other analyses require retrieving the traveled paths of a large set of trajectories between two points.

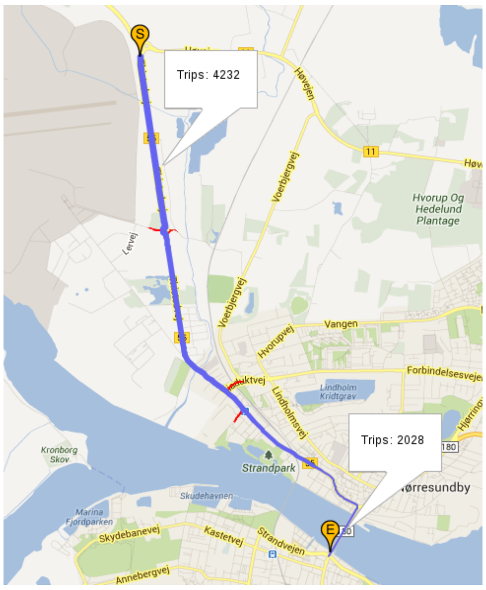

Figure 1: The "Napoleon campaign".

One example analysis where the traveled path is of importance is Figure 1 that visualizes how traffic moves along a main traffic artery. The width of the blue line denotes the number of trajectories following the path from 'S' to 'E', and red lines highlight frequent exits. The image provides concrete knowledge on where and how to optimize, e.g., the signalized intersections and turn-lanes between the start and end point. Figure 1 resembles the image depicting Napoleon's Russian campaign, by Charles Joseph Minard from 1861 (see [8] for more information), and has been discussed frequently in spatial data-mining context [3].

Figure 1 can be created by retrieving the trajectories following each of the sub-paths from the beginning to the end, i.e., if the path consists of the edges $< e_1, e_2, e_3, \ldots, e_n >$ retrieve the trajectories following $< e_1 >, < e_1, e_2 >, < e_1, e_2, e_3 >, \ldots, < e_1, e_2, e_3, \ldots, e_n >$.

Another application where the specific path is of significance is evaluating the measures proposed in [7] for analysis of traffic conditions using trajectories. The proposed measures (travel time, congestion index, proportion stopped time, and acceleration noise) provide unique insights into the traffic and infrastructural conditions on a given path. Intrinsically, evaluating any of these measures requires retrieving the trajectories that follow a specific path in the road network. Furthermore, an important application for traffic planners is to find the traveled paths between two points and determine the extent to which vehicles follow desired paths. For instance it is preferable that traffic avoids schools of safety reasons, but determining whether this is the case is impossible with point-based measurements.

The Interactive Traffic Analysis (INTRA) system answers these queries. INTRA contains a simple index, that enables efficient retrieval of trajectories following a specific path in the road network. The system can be used to determine whether two trajectories follow the exact same path between two points (without retrieving the full paths). Response times are in general below one second, even for our very large real-world data set containing 1.5 million trajectories, which enables us to evaluate the traffic analyses interactively.

In the remainder of this demonstration proposal we describe the core aspects of the INTRA system and present the data set used. Finally, we describe what we intend to demonstrate and conclude.

2. THE INTRA SYSTEM

INTRA has a client-server architecture as shown on Figure 2. The client is a web-interface implemented in PHP 5.4 and JavaScript. The server stores and queries the trajectory data. The client communicates with the server through a REST API. The Query Processing component in the server is implemented using Python 3.2, whereas the Trajectory Store and OSM components are implemented in PostgreSQL 9.2 with the PostGIS 2.0 extension. The OSM component contains the road-network imported from the OpenStreetMap project [1]. The Trajectory Store component indexes all trajectories and the Query Processing layer contains the necessary logic to evaluate the queries.

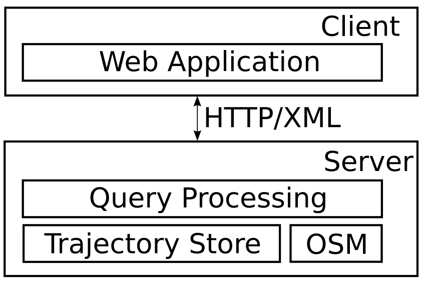

Figure 2: The INTRA architecture.

The Trajectory Store and Query Processing components are of most interest and are described in details in the following sections.

2.1 Trajectory Store

The trajectory store takes direct outset in state-of-the-art in network constrained indexing, [2, 5, 6], where movement is represented as 4-tuples, $(tid, eid, t_{start}, t_{end})$. tid is a trajectory identifier, eid an edge identifier, t_{start} and t_{end} the time of the first and last GPS record on the edge eid. Our addition is a hash attribute computed by applying a hash function to the list of edges touched by the trajectory. The idea of using a hash attribute, is that its evolution between two points depends on the specific path followed.

Equation 1 defines our hash function. The $path$ is the list of edges touched by a trajectory up to time t_{end}. $e.weight$ is a weight assigned to each edge. The weight is discussed further in Section 2.3.

$$hash(path) = \sum_{e \in path} e.weight \qquad (1)$$

Figure 3 shows example input data. Each GPS record is map-matched to the nearest edge and the route followed by each trajectory is computed. Table 1 shows the result of loading the trajectories shown in Figure 3. Note that the pentagon (green) trajectory has an entry for edge 3 even though no GPS records are matched to this edge. This is because the $path$ of each trajectory is stored, and not only edges with GPS records. The $hash$ attribute in Table 1 is computed by setting $e.weight = e.id$. For instance, the circle (red) trajectory touches edges 1, 3 and 5, and its corresponding hash values are therefore $hash(<1>) = 1$, $hash(<1,3>) = 4$, and $hash(<1,3,5>) = 9$.

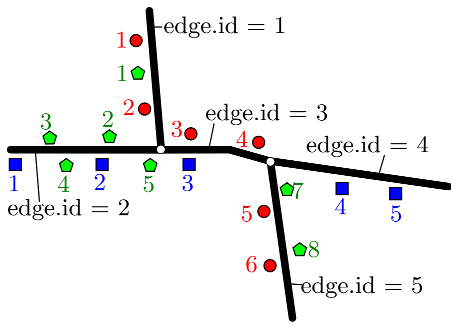

Figure 3: Example map and trajectories.

tid	eid	t_{enter}	t_{leave}	$hash$
🟦	2	1	2	2
🟦	3	3	3	5
🟦	4	4	5	9
🔴	1	1	2	1
🔴	3	3	4	4
🔴	5	5	6	9
⬟	1	1	1	1
⬟	2	2	5	3
⬟	3	5	7	6
⬟	5	7	8	11

Table 1: The `visitedsegments` table.

A covering B$^+$tree is added on the attributes eid, tid, t_{enter}, t_{leave}, and $hash$. The order of the first two attributes is important, to allow filtering on spatial and temporal constraints as described further in the following section. Since this indexing scheme only requires a B$^+$tree, it integrates easily with existing database management systems.

```
--Retrieve trajectories following path
select v1.tid, v1.tenter, v2.tleave
from   visitedsegments as v1
join   visitedsegments as v2 using (tid)
where  v1.eid = head(querypath)
and    v2.eid = last(querypath)
and    v2.hash = v1.hash + deltahash(querypath)

--Retrieve one trajectory per path
select max(v1.tid) as tid
from   visitedsegments as v1
join   visitedsegments as v2 using (tid)
where  v1.eid = firstedge
and    v2.eid = lastedge
and    v1.hash < v2.hash
group by (v2.hash - v1.hash)
```

Figure 4: Strict path query execution.

2.2 Query Processing

We describe how to find all trajectories that strictly follow a given path, and how to retrieve one trajectory per path between two points. Our algorithm for executing a strict path query starts by retrieving the trajectories that touch the first and last edge in the path, and then removes the trajectories where the evolution in the hash attribute between the first and last edge does not exactly match the evolution determined by the queried path. The evolution of the *hash* attribute is computed using the *deltahash* function, defined as $deltahash(path) = hash(tail(path))$, where the *tail* function removes the first edge in the path. In other words, $deltahash(path)$ describes exactly the change in the hash attribute between entering the path and leaving it for all trajectories in the result set.

Consider the example in Figure 3 and the strict path query $= < 3, 5 >$. First the trajectories touching edge 3 and edge 5 are retrieved, i.e., the circle and pentagon trajectories. Then the evolution in the hash attribute between entering edge 3 and leaving edge 5 is verified to match the evolution determined by the queried path, i.e., for the circle trajectory we check that $4 + deltahash(< 3, 5 >) = 9$, and for the pentagon we check that $6 + deltahash(< 3, 5 >) = 11$. Both are true, thus both trajectories are returned.

This strategy can be implemented in SQL, as shown in Figure 4 (top). The variable *querypath* contains the path in the strict path query and the two functions *head* and *last* returns the first and last edge of *querypath*, respectively.

To retrieve all traveled paths between two edges, $firstedge$ and *lastedge*, the trajectories touching both edges are retrieved. Then one trajectory identifier is returned for each unique delta. Finally, for each trajectory identifier returned, the path is retrieved. Other network constrained indexing techniques needs to retrieve the path of each trajectory touching both edges. Retrieving the path of a trajectory usually incurs one random IO per trajectory, and is thus expensive for a large trajectory data set. INTRA has a significant advantage for this type of query in that the number of unique paths followed between two points is usually much lower than the number of trajectories in-between. For instance see Figure 7, where more than 300 trajectories follow 10 unique paths (some paths are hidden beneath other paths). In this case, INTRA needs to retrieve the path of 11 trajectories, whereas other approaches needs to retrieve more than 300.

Figure 4 (bottom) shows the SQL for identifying one trajectory per traveled path. After executing this query, the path of the resulting trajectories needs to be retrieved (a trivial select statement, omitted due to space considerations). A simple extension of this query is to compute the average travel time on each path or the number of trajectories following each path, and only return the 10 fastest/most frequent paths.

Both queries in Figure 4 will use the covering B$^+$tree described in Section 2.1.

2.3 Edge Weights

The queries shown in Figure 4 may give incorrect results when $deltahash(\pi_1) = deltahash(\pi_2) \wedge head(\pi_1) = head(\pi_2) \wedge last(\pi_1) = last(\pi_2) \wedge \pi_1 \neq \pi_2$, for two paths π_1 and π_2. However, by carefully choosing the weight, $e.weight$, for each edge the probability of incorrect results can be made extremely low.

Concretely, each edge is assigned a weight as defined by Equation 2. The $prime(n)$ function returns the n'th prime number. The intuition of these weights builds directly on the fundamental theorem of arithmetic [4], which states that the product of a unique set of primes will be unique. We use this to our advantage by assigning a unique prime number to each edge, and use the unique product of primes along the path as hash function. Applying the logarithm transformation preserves uniqueness, and allows us to rewrite: $\log \prod_{p \in path} p$, into: $\sum_{p \in path} \log p$, which is compatible with our hash function as defined in Equation 1.

$$e.weight = \log prime(e.id) \tag{2}$$

We store the hash attribute in an 8-byte fixed-point field that store 12 decimals. Using this representation we have executed more than 8 million strict path queries (of various lengths) and verified the results. We have not identified a single instance where this approach yields incorrect results, and therefore find that this source of error is insignificant to other sources such as map-matching and location error.

3. DEMONSTRATION

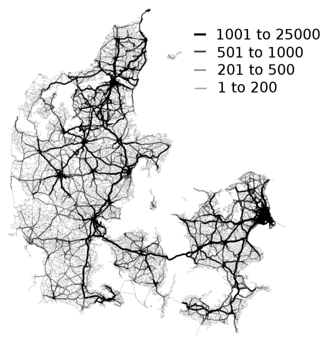

Figure 5: Trajectories per edge in our data-set.

The web-site `daisy.aau.dk/its/spqdemo/`, will be used in the demonstration, and contains example video of the three use cases to be demonstrated. All three use cases are based on input from traffic researchers.

A very large real-world data set consisting of 1.9 billion GPS records is used. Preprocessing results in 50 million en-

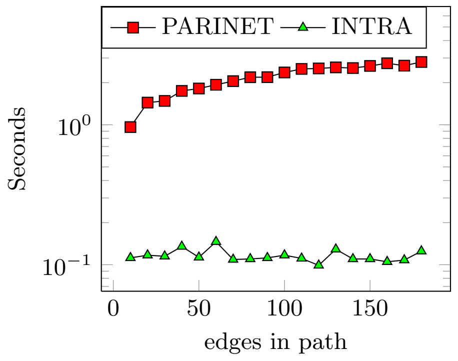

Figure 6: Response times of strict path queries.

tries such as those shown in Table 1. The road network used is from OpenStreetMap, and consists of 659 595 edges. The data set is visualized in Figure 5, and shows good coverage in urban areas, with up to 25 000 trajectories touching each edge.

Low query response times are required for an interactive demonstration. Comparing the query performance of our representation with state-of-the-art, i.e., PARINET [6], we see that the query response time of our representation is usually below one second and significantly below the response time of PARINET for strict path query evaluation.

Use case 1: Retrieve the trajectories that follow a specific path in the road network, e.g., the full path highlighted in Figure 1. The trajectories can be retrieved with different temporal predicates, e.g., all trajectories on the path within a time interval, or all trajectories on Mondays between 8 and 9 AM, or a combination of these two. For selected paths, we show measures such as travel-time, and perform fine-grained analysis of, e.g., how the speed evolved for each trajectory along the path. This use case is important for determining whether signalized intersections are coordinated properly, or when computing average turn-times for intersections.

Use case 2: Answer the query: "given two points, what routes do trajectories follow in between". Figure 7 shows this use case. 'S' and 'E' marks the two points, and the thickness of each line indicates the number of trajectories that follow it. The thin red line shows the fastest route from Google Maps. The figure enables us to determine the fastest or most frequent routes between the points, and compare this with the fastest route provided by Google. This information is valuable for, e.g., deciding what route to follow or how to optimize traffic flows. For instance, with this figure we observe that 167 trajectories follow a longer and slower route, and that 121 trajectories follow the shorter and faster route. Furthermore, we observe that the travel time estimate of the path suggested by Google, is 189 seconds, whereas the actual average travel time of the fastest route is 178 seconds.

Use case 3: Determine the traffic flow along a main artery. As input we select a path along a traffic artery. For each edge in the path, the number of vehicles following the path up to that edge is found by executing a strict path query. When vehicles leave the path, the exit edge of the leaving vehicles is found and illustrated as well. This helps the traffic planner to, e.g., identify the turn-lanes that should be optimized along the traffic artery. An example output of this is shown on Figure 1.

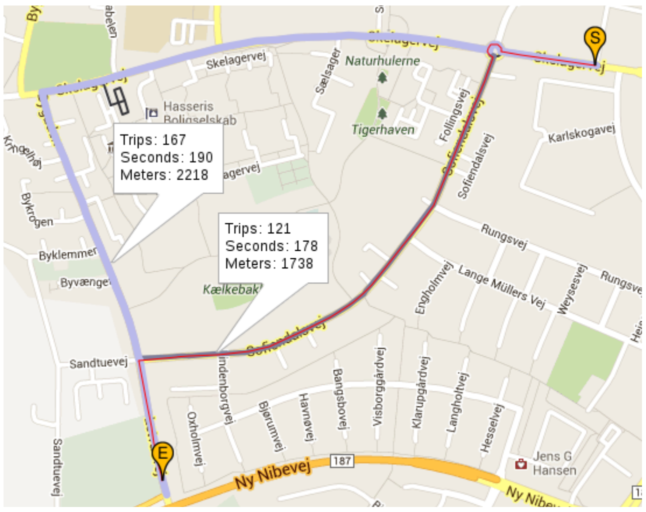

Figure 7: Traveled paths between 'S' and 'E'.

4. CONCLUSION

We have presented the INTRA system, which enables interactive web-based point-and-click traffic analysis. Three different use cases are presented; one for analyzing a specific path, one for analyzing the different routes between two endpoints, and one visualizing the flow along a traffic artery.

Acknowledgments

Authors from Aalborg University are funded by the EU FP7 REDUCTION project (`www.reduction-project.eu`). Authors from University of Piraeus are funded by EU FP7 DATASIM project (`www.datasim-fp7.eu`)

5. REFERENCES

[1] OpenStreetMap. `http://www.openstreetmap.org/`. Retrieved Apr. 30, 2013.

[2] V. T. de Almeida and R. H. Güting. Indexing the trajectories of moving objects in networks. *GeoInformatica*, 9(1):33–60, 2005.

[3] F. Giannotti and D. Pedreschi. Mobility, data mining and privacy: A vision of convergence. In *Mobility, Data Mining and Privacy*, pages 1–11. Springer, 2008.

[4] G. Hardy and E. Wright. *An introduction to the theory of numbers*. Oxford University Press, USA, 1980.

[5] D. Pfoser and C. S. Jensen. Indexing of network constrained moving objects. In *ACM-GIS*, pages 25–32. ACM, 2003.

[6] I. S. Popa, K. Zeitouni, V. Oria, D. Barth, and S. Vial. Indexing in-network trajectory flows. *VLDB J.*, 20(5):643–669, 2011.

[7] M. Taylor, J. Woolley, and R. Zito. Integration of the global positioning system and geographical information systems for traffic congestion studies. *Transportation Research Part C: Emerging Technologies*, 8(1-6):257–285, 2000.

[8] E. R. Tufte and P. Graves-Morris. *The visual display of quantitative information*, volume 2. Graphics press Cheshire, CT, 1983.