Efficient One-click Browsing of Large Trajectory Sets

Benjamin Krogh, Ove Andersen, Edwin Lewis-Kelham, Kristian Torp
Dept. of Computer Science
Aalborg University, Denmark
{ bkrogh, xcalibur, edwin, torp }@cs.aau.dk

ABSTRACT

Traffic researchers, planners, and analysts want a simple way to query the large quantities of GPS trajectories collected from vehicles. In addition, users expect the results to be presented immediately even when querying very large transportation networks with huge trajectory data sets. This paper presents a novel query type called *sheaf*, where users can browse trajectory data sets using a single mouse click. Sheaves are very versatile and can be used for location-based advertising, travel-time analysis, intersection analysis, and reachability analysis (isochrones). A novel in-memory trajectory index compresses the data by a factor of 12.4 and enables execution of sheaf queries in 40 ms. This is up to 2 orders of magnitude faster than existing work. We demonstrate the simplicity, versatility, and efficiency of sheaf queries using a real-world trajectory set consisting of 2.7 million trajectories (1.36 billion GPS records) and a network with 1.5 million edges.

Categories and Subject Descriptors

H.2.8 [**Database Applications**]: Spatial databases and GIS

Keywords

trajectories, traffic-analysis, moving-objects

1. INTRODUCTION

The proliferation of GPS-enabled devices continues to challenge researchers to develop new methods for analyzing large trajectory data sets. A major challenge is creating visualizations that are both scalable, easy to interpret, and usable for multiple purposes.

In this demo, we present a novel system for evaluating and visualizing *sheaf* queries, on network constrained trajectory data. The sheaf query is conceptually simple, and provides an efficient way of browsing large trajectory data sets.

SIGSPATIAL '14, Nov 04-07 2014, Dallas/Fort Worth, TX, USA
ACM 978-1-4503-3131-9/14/11.
http://dx.doi.org/10.1145/2666310.2666371

The idea is that the user selects a *sheaf center*, i.e., a directed edge in the transportation network. Trajectories that visit this center are then retrieved and visualized. Figure 1 illustrates this concept. The solid lines show a small (directed) transportation network with a user selected sheaf center and three trajectories that touch the sheaf center. Each edge has a visit count, i.e., the number of times one of these trajectories touch it. The sheaf then consists of all edges with a non-zero visit count. A sheaf has an *incoming* and an *outgoing* half. The incoming half consists of the edges touched by trajectories before visiting the sheaf center and the *outgoing* half consists of the edges touched after visiting the sheaf center. The gray areas in Figure 1 show the two sheaf halves.

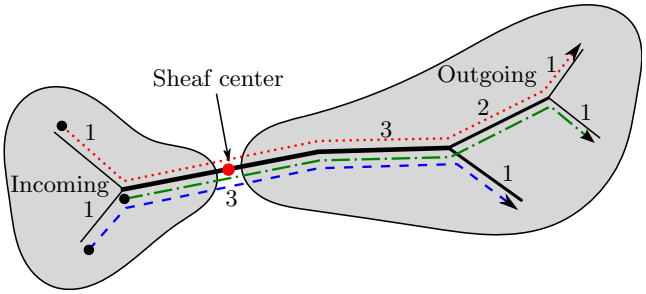

Figure 1: The Sheaf Concept

Sheaves has a plethora of use cases, such as analysis of location-based advertising, travel-times, congestion impact, route choice behavior, and traffic optimization. Further, sheaves are useful on both a regional (major roads) and on a local scale (single intersection). The conceptual idea of sheaves is from the traffic literature, but this is to the best of our knowledge the first system capable of efficiently evaluating sheaf queries on a large set of trajectories.

The system demonstrated supports sophisticated temporal filtering using any combination of time of day, day of week, month of year, and within a date interval (see Section 3.2). This filtering enables comparative analysis of peak hour and non-peak hour traffic.

A novel and efficient in-memory index speeds up sheaf evaluation by up to two orders of magnitude over baseline methods. Evaluating a sheaf query takes less than 40 ms on average. The index uses state-of-the-art compression [6], which reduces memory requirements by a factor of 12.4, from 1494 MB, to 120.2 MB. The demonstration uses a very large trajectory data set, containing 2.7 million trajectories (1.36 billion GPS records), and a transportation network consisting of 1.5 million edges.

2. RELATED WORK

The iRoad system [4] is based on the reachability tree, which is a tree that includes all locations that can be reached within a temporal window, from a user-defined root. Sheaves resemble reachability trees, but are more expressive in that they show the actual movement of objects, whereas reachability trees show the predicted movement.

Compression strategies for network constrained trajectory data are discussed in [6]. The authors find that the best compression is achieved using Shortest Path Encoding (SPE) (see Section 3.3). The authors conclude that SPE is of limited applicability because decoding is too expensive.

We adapt and integrate SPE with the Hub Labeling (HL) algorithm [1]. HL uses less than one μs to compute a shortest path, which enables efficient decoding of SPE encoded trajectories and makes SPE viable for real-world applications.

3. TECHNICAL BACKGROUND

The transportation network is modeled as a graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$, where $\mathbf{V}$ is a set of vertices and $\mathbf{E}$ is a set of directed edges. Each edge, $e \in \mathbf{E}$, starts at the vertex $startvertex(e)$ and ends at the vertex $endvertex(e)$.

The position updates from moving objects are cleansed, map-matched, and integrated into the data warehouse described in [2]. The *trajectory* of a moving object describes the historical movement of the object. For instance, driving to work or to the mall is a single trajectory. The path of a trajectory, $\pi = [e_1, e_2, \ldots, e_n]$, $e_i \in \mathbf{E}$, is represented by an ordered list of edges, where edge e_i is visited before edge e_{i+1}. The temporal evolution of a trajectory is represented by a list of timestamps, $[ts_1, ts_2, \ldots, ts_n]$, where ts_i is the time at which the moving object is at the vertex $v_i = endvertex(e_i)$. ts_i is linearly interpolated between the last position update before visiting v_i and the first position update after visiting v_i.

3.1 Sheaf Definitions

The sheaf query is formalized in Equation 1. The result consists of all edges from trajectories that touch the sheaf center on edge e_c. Each edge, e, in the sheaf must be on the path, π_t of a trajectory $t \in \mathbf{T}$, where $\mathbf{T}$ is the set of trajectories that touch e_c. Further, the *spatial* network distance between e_c and e along π_t, denoted by $dist_{net}(e_c, e, \pi_t)$, must be less than the r. The *temporal* distance, $dist_{time}(e_c, e, t)$, between e_c and e must be less than the period p.

$$sheaf(e_c, r, p, \mathbf{T}) = \{e | e \in \pi_t \wedge t \in \mathbf{T} \wedge dist_{net}(e_c, e, \pi_t) < r \wedge dist_{time}(e_c, e, t) < p\} \quad (1)$$

The r parameter is used to configure how *local* the sheaf should be. For instance when analyzing an intersection, this parameter can be used to limit the number of times a trajectory can pass the intersection. The p parameter is useful for reachability analysis, i.e., showing the edges that can be reached from the sheaf center within a specific period p.

In some cases it is difficult to interpret the paths to/from the sheaf center. The *sheaf-tree* improves this, by only showing the dominating paths to/from the sheaf center. Each edge, e, in a sheaf-tree has exactly one parent edge, $parent(e)$. The root edge that contains the sheaf center does not have any parent. The sheaf-tree is computed using a breadth-first search from the root of the tree that incrementally grows the most frequent branches of the tree. If two branches intersect at some point, only the most frequent branch continues. This process is similar to the Incremental Network Expansion algorithm in [7]. The sheaf-tree is formalized by Equation 2. The function $prev(e, \pi_t)$ returns the edge in π_t prior to e.

$$sheaf\text{-}tree(e_c, r, p, \mathbf{T}) = \{e | e \in \pi_t \wedge t \in \mathbf{T} \wedge dist_{net}(e_c, e, \pi_t) < r \wedge dist_{time}(e_c, e, t) < p \wedge prev(e, \pi_t) = parent(e)\} \quad (2)$$

3.2 Sheaf Evaluation

In order to evaluate a sheaf query, the trajectory data needs to be identified and retrieved. Two steps are used for this. First, the set of trajectories, $\mathbf{T}$, that touch e is identified. Next, the full trajectory data for each trajectory $t \in \mathbf{T}$ is retrieved. After these two steps, the remaining processing is trivial aggregation.

The table design in Table 1 is used to create $\mathbf{T}$. A composite B$^+$-tree index on the columns *eid*, $time_{enter}$, $time_{leave}$, and *tid* (in this order) enables efficient retrieval of the trajectory identifiers that touch the edge containing the sheaf center. Further, it is possible to apply temporal filters using this index. Concretely, the set of trajectories $\mathbf{T}$ can be constrained such that each trajectory touch e within a specified time of day, day of week, month of year, and within a temporal interval. For instance, it is possible to only include trajectories between 7:00 and 9:00 AM on Mondays, in January or February, in the interval 2007 to 2008.

Column	Description
tid	The trajectory identifier
eid	The edge identifier
$time_{enter}$	The time trajectory *tid* entered *eid*
$time_{leave}$	The time trajectory *tid* left edge *eid*

Table 1: On Disk Format for Trajectories

Retrieving the full trajectory data is more challenging. State-of-the-art in network-constrained indexing, [8], focus on retrieving the trajectories within a query range and cannot retrieve individual trajectories efficiently.

Consider using a B$^+$-tree, $B_{BASELINE}$, on the columns *tid*, *eid*, $time_{enter}$, and $time_{leave}$ for retrieving trajectories. Then each trajectory in $\mathbf{T}$ will, in the worst case, require one random I/O. Since $\mathbf{T}$ can be very large (tens of thousands), this is very inefficient. A new index for retrieving trajectory data is therefore required, see Section 3.3.

3.3 Trajectory Compression

Efficient retrieval of trajectories is essential for the evaluation of sheaf queries. This retrieval is accelerated by compressing the trajectories and storing them in main memory. The SPE path encoding is suggested in [6]. The intuition behind SPE is that moving objects usually follow the shortest paths between edges in a transportation network. A compact representation is therefore to store only the (few) edges that need to be connected with the shortest path. SPE is lossless, and can represent any path through the transportation network. SPE is performed using the algorithm from [6], modified to use HL for shortest path computations.

The two paths, π_{dash} and π_{dot}, in Figure 2 illustrate the SPE encoding. In the figure, vertices and edges are rep-

resented by circles and solid lines, respectively. There is a directed edge, $e_{1,2}$, between vertex 1 and vertex 2 if the two vertices are connected with a solid line.

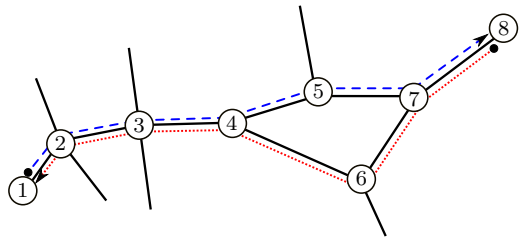

Figure 2: Two Paths in a Transportation Network

The dashed and dotted paths are represented by the edges $\pi_{dash} = [e_{1,2}, e_{2,3}, e_{3,4}, e_{4,5}, e_{5,7}, e_{7,8}]$ and $\pi_{dot} = [e_{8,7}, e_{7,6}, e_{6,4}, e_{4,3}, e_{3,2}, e_{2,1}]$, respectively. By examining the transportation network in Figure 2, we observe that the shortest path from $e_{1,2}$ to $e_{7,8}$ match exactly π_{dash} between these two edges. The SPE encoding of π_{dash} is therefore $SPE(\pi_{dash}) = [e_{1,2}, e_{7,8}]$. SPE is the function used to encode the path.

The longest shortest-path from $e_{8,7}$ along π_{dot} ends on $e_{6,4}$. Because $e_{6,4}$ is not the last edge in π_{dot} another shortest path from $e_{6,4}$ to $e_{2,1}$ is required. The encoding of π_{dot} is therefore $SPE(\pi_{dot}) = [e_{8,7}, e_{6,4}, e_{2,1}]$. Table 2 shows the original and SPE encoded paths. A trajectory in our data set touch on average 69 edges, which is reduced to 4 edges by SPE. The first row in Table 3 shows the space savings for the 2.7 million trajectories in our data set.

Id	Original path	SPE coded
π_{dash}	$[e_{1,2}, e_{2,3}, e_{3,4}, e_{4,5}, e_{5,7}, e_{7,8}]$	$[e_{1,2}, e_{7,8}]$
π_{dot}	$[e_{8,7}, e_{7,6}, e_{6,4}, e_{4,3}, e_{3,2}, e_{2,1}]$	$[e_{8,7}, e_{6,4}, e_{2,1}]$

Table 2: SPE Coding of Trajectory Paths

Restoring the original path from an SPE encoded list of edges is relatively simple. For each pair of consecutive edges, (e_i, e_{i+1}) in an SPE encoded path, e_i and e_{i+1} are connected using the shortest path. Concretely, to decode the encoded path π_{dash}, it is necessary to compute the shortest path between $endvertex(e_{1,2})$ and $startvertex(e_{7,8})$. This shortest path is $[e_{2,3}, e_{3,4}, e_{4,5}, e_{5,7}]$. The restored path is therefore $[e_{1,2}, e_{2,3}, e_{3,4}, e_{4,5}, e_{5,7}, e_{7,8}]$, i.e., the original path, π_{dash}.

Thousands of trajectories may be included in a sheaf, which implies thousands of shortest-path computations. For instance, approx. 27 750 shortest paths are used to represent the paths of the 9250 trajectories in Figure 3. Decoding these trajectories using HL takes less than 60 ms.

The temporal information of trajectories is also compressed. The temporal information of a trajectory is a list of increasing integers, with one value per edge in the path. Such a list is compressed using the technique described in [5]. On the data set used, this reduces the size by a factor of 4, from 198 million 4-byte integers to 47.2 million. By reducing the granularity from 1 second, to 30 seconds, the deltas become smaller, and significantly improves the compression. This further reduces the space consumption to 13.4 million integers, i.e., a reduction of one order of magnitude. Table 3 shows the summarized space savings by the spatial and temporal compression.

Encoding the spatial and temporal information of a trajectory is very efficient using this technical setup. More than 30 000 trajectories can be encoded per second using a desktop Intel i7-3770 CPU. As such, adding new trajectories to the in-memory index is very efficient. The compressed information is stored in a list in memory, with a start and end offset for each trajectory. Finally, Table 4 compares the performance of the SPE index, and $B_{BASELINE}$ for evaluating the sheaves shown in Section 4.

Data type	Original	Compressed	Ratio
Spatial	742 MB	44.8 MB	16.6
Temporal	752 MB	75.4 MB	10.0

Table 3: Spatial and Temporal Compression

Implementation	Figure 3	Figure 4	Figure 5
$B_{BASELINE}$	39 796 ms	26 505 ms	29 938 ms
SPE index	60 ms	39 ms	40 ms

Table 4: Data Retrieval, SPE Index, and $B_{BASELINE}$

4. DEMONSTRATION

We will demonstrate a number of scenarios using a very large real-world trajectory data set consisting of 2.7 million trajectories (1.36 billion GPS records). A web-based interface is used, and we invite the audience to experiment with sheaves using this interface. All screenshots use Google Maps.

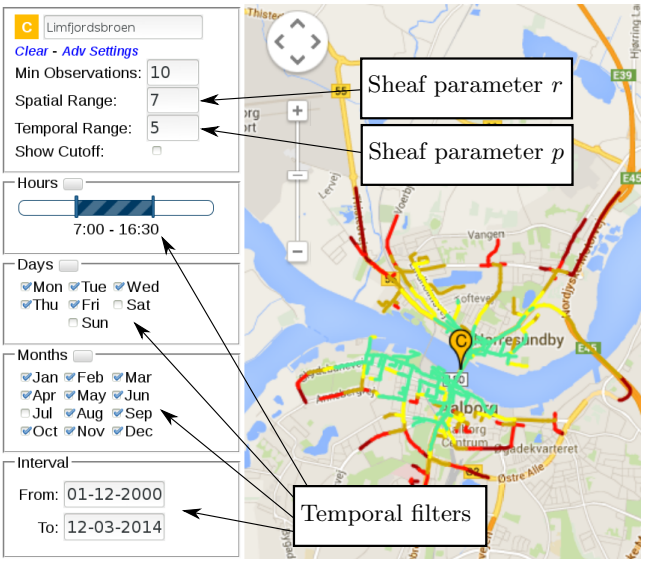

Figure 3: Isochrone from Bridge, 9250 Trajectories

4.1 Isochrones

In the first scenario the user wants to examine how far drivers can get within five minutes from the sheaf center, i.e., the 5 minute isochrone [3]. The screenshot in Figure 3 shows such a sheaf. The sheaf center is placed at a bridge, which makes the sheaf shape more clear. A five minute temporal range is specified, and the green, yellow, and red parts are reached within this limit. Specifically, green edges are reached by [90-100]% of trajectories, yellow edges by [50-90)%, and red edges by (0-50)%. Edges not reached by any trajectory within the temporal period are here not included in the sheaf.

Figure 3 also shows the temporal filter time-of-day is between 7:00 and 16:30 on weekdays on all months of the year except July (summer vacation) in the interval from 2000 to 2014. The demonstration will show how the isochrones change due to traffic congestion. Further, we will use the sheaf query to find the regions that are within 10, 15, and 20 minutes driving from a hospital. Finally, the bridge example will be used to demonstrate how easy the sheaf-tree is to interpret compared to the regular sheaf.

4.2 Incoming/outgoing Analysis

In traffic analysis it is of interest where vehicles come from and are going to. An example of such incoming/outgoing trajectory analysis is shown in Figure 4, where the sheaf center is placed between two roundabouts. The trajectories shown are on workdays, between 7:00 and 9:00. The blue-colored lines to the left and the red-colored lines to the right show the incoming and outgoing half of a sheaf, respectively. Further, the thickness of each edge indicates the number of trajectories that touch it. From Figure 4, it is immediately clear that most trajectories pass straight through the left roundabout and take the second exit in the right roundabout. Note that the left roundabout has incoming traffic from only two directions and that the five legged roundabout to the right has trajectories in all four outgoing directions.

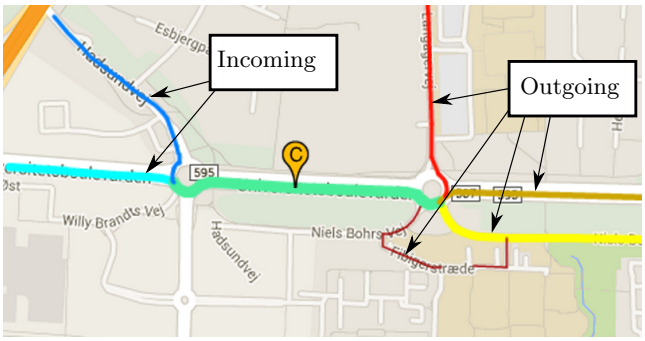

Figure 4: Incoming/Outgoing, 3678 Trajectories

The demonstration will show how a shopping center can use sheaves to determine where trajectories that pass the center are coming from and going to. Such information is highly relevant for location-based advertising. The temporal filters can be used for comparing morning and afternoon rush hours or for comparing last year's trajectories with this year's trajectories.

4.3 Intersection Analysis

Signalized intersections regulate the flow of traffic in cities and expensive equipment is often used to monitor the most vital intersections. Using sheaves, it is possible to monitor all intersections without additional equipment.

An analysis of a major intersection is shown in Figure 5. The figure shows the outgoing half of a sheaf, with the center in the green line, for weekdays between 7:00 and 9:00 from 1st of January 2012 and forward. Again, the width of the lines indicates how many trajectories go in each direction. For this particular intersection, 63% of the trajectories turn left, 19% turn right, and only 16% go straight. 2% follow other directions, e.g., take a U-turn or ends. This information is very relevant for optimizing the traffic signals to match the actual traffic.

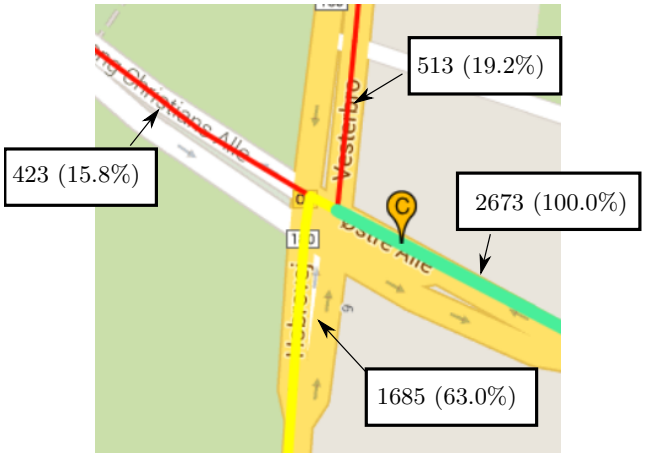

Figure 5: Intersection Analysis, 2673 Trajectories

In the demonstration, we will show how sheaves can be used to analyze how intersection usage changes throughout the day. Further, we will also show how sheaves can be used to identify frequently used exits along a motorway. This is very relevant for traffic analysts.

5. SUMMARY

The demonstration shows how sheaves can be used for a wide range of purposes including traffic analysis, location-based advertising, and travel time analysis. A novel and highly efficient technical setup enables random access to the spatio-temporal information of each trajectory, and allows users to interactively browse very large trajectory sets, using only a single mouse click.

Acknowledgment

This work is supported by the REDUCTION project `www.reduction-project.eu`.

6. REFERENCES

[1] I. Abraham, D. Delling, A. V. Goldberg, and R. F. F. Werneck. A hub-based labeling algorithm for shortest paths in road networks. In *SEA*, pages 230–241, 2011.

[2] O. Andersen, B. B. Krogh, and K. Torp. An open-source based ITS platform. In *MDM*, volume 2, pages 27–32. IEEE, 2013.

[3] J. Gamper, M. Böhlen, and M. Innerebner. Scalable computation of isochrones with network expiration. In *SSDBM*, pages 526–543. Springer, 2012.

[4] A. M. Hendawi, J. Bao, and M. F. Mokbel. iRoad: a framework for scalable predictive query processing on road networks. *PVLDB*, 6(12):1262–1265, 2013.

[5] D. Lemire and L. Boytsov. Decoding billions of integers per second through vectorization. *Software: Practice and Experience*, 2013.

[6] P. M. Lerin, D. Yamamoto, and N. Takahashi. Encoding network–constrained travel trajectories using routing algorithms. *Int. J. Knowledge and Web Intelligence*, 4(1):34–49, 2013.

[7] D. Papadias, J. Zhang, N. Mamoulis, and Y. Tao. Query processing in spatial network databases. In *PVLDB*, pages 802–813. VLDB Endowment, 2003.

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