EasyChair Preprint

№ 1409

TraV: an Interactive Trajectory Exploration System for Massive Data Sets

Jieliang Ang, Tianyuan Fu, Johns Paul, Shuhao Zhang, Bingsheng He, Teddy Sison David Wenceslao and Sienyi Tan

EasyChair preprints are intended for rapid dissemination of research results and are integrated with the rest of EasyChair.

August 24, 2019

TraV: AN INTERACTIVE TRAJECTORY EXPLORATION SYSTEM FOR MASSIVE DATA SETS

Jieliang Ang, Tianyuan Fu, Johns Paul, Shuhao Zhang, Bingsheng He
Teddy Sison David Wenceslao, Sien Yi Tan

Grab-NUS AI Lab

ABSTRACT

The proliferation of modern GPS-enabled devices like smartphones have led to significant research interest in *large-scale trajectory exploration*, which aims to identify all nearby trajectories of a given input trajectory. Trajectory exploration is useful in many scenarios, for example, in identifying incorrect road network information or in assisting users when traveling in unfamiliar geographical regions as it can reveal the popularity of certain routes/trajectories. In this study, we develop an interactive trajectory exploration system, named TraV. TraV allows users to easily plot and explore trajectories using an interactive Graphical User Interface (GUI) containing a map of the geographical region. TraV applies the Hidden Markov Model to calibrate the user input trajectory and then makes use of the massively parallel execution capabilities of modern hardware to quickly identify nearby trajectories to the input provided by the user. In order to ensure a seamless user experience, TraV adopts a progressive execution model that contrasts to the conventional "query-before-process" model. Demonstration participants will gain experience with TraV and its ability to calibrate user input and analyze billions of trajectories obtained from Grab drivers in Singapore.

1. INTRODUCTION

Due to the widespread usage of GPS-enabled devices like smartphones, massive amounts of trajectory data is being generated in real time. Analysing such trajectory data is critical to a number of domains including transport analysis, animal movement study etc. In this work, we study the problem of *large scale trajectory exploration*, which aims to identify all nearby trajectories of a given input trajectory.

Such a query is fundamentally different from related works on trajectory similarity search [12] or trajectory similarity joins [11] as the *nearby* trajectories are not necessarily *similar* to the input trajectory as a whole but they may partially pass by the same area of interest. In Grab [3], we have found this query to be especially beneficial. For example, it can help identify incorrect road network information (e.g. missing roads) and assist users travelling to unfamiliar geographical regions (e.g., identify popular nearby trajectories). However in-spite of its usefulness, supporting such a query is challenging due to the following reasons.

First, composing an input trajectory itself is often a tedious task. We found that data scientists often spend considerable amounts of time in expressing their trajectory queries as precise geographical coordinates, either through SQL or other general purpose programming languages (e.g., Java, Python). Second, the high computational complexity makes the naive implementation of this query impractical. Specifically, given an input trajectory, the naive implementation decomposes the input into a list of <start, end> coordinates. The implementation then iterates through all relevant historical trajectories to identify trajectories that are close enough (determined based on predefined threshold) to every pair of <start, end> positions, leading to high computational complexity. Third, a trajectory exploration system should ensure fast response to user's query, especially when it is intended for use by non-technical users for the purposes of identifying interesting trajectories in an unfamiliar geographical region. Given the high computational cost of trajectory exploration, this is almost impossible to achieve when adopting the conventional "query-before-process" execution model that requires the user to express the entire input trajectory before the trajectory exploration can be initiated.

To address the above challenges, we develop an interactive trajectory exploration system named TraV. To address the first challenge, TraV allows users to express their input trajectory using a simple Graphical User Interface (GUI). However, such a GUI based implementation comes with its own sets of challenges. For example, plotting trajectories by hand is hardly accurate. Hence, TraV calibrates the input automatically and progressively based on the Hidden Markov Model when the users are expressing their trajectory. This automatic calibration is analogous to automatic text correction. To handle the second challenge, TraV makes use of aggressive indexing and takes advantage of the massive parallelism provided by modern hardware devices. The use of indexing allows TraV to minimize the amount of computation and the parallel execution model allows it to concurrently process massive amounts of underlying trajectory data. To address the

Jieliang Ang and Tianyuan Fu contribute equally to this work. The corresponding authors are Johns Paul and Shuhao Zhang.

third challenge, TraV adopts a progressive execution model that calibrates the input trajectories and explores the underlying trajectory data while the input trajectory data is being expressed by the user. This provides quick responses to the user, thus ensuring a seamless user experience.

Overall, our demonstration of TraV aims to show the usefulness of an interactive trajectory exploration system capable of exploring billions of GPS coordinates in real-time. In the remainder of this paper, we present the implementation and describe our solutions.

2. TRAJECTORY EXPLORATION

Trajectory exploration aims to identify nearby trajectories of an input trajectory. In the typical scenarios we explored in this study (i.e., identifying incorrect road network data and assisting users in exploring unfamiliar geographical regions), what we are interested in are the trajectories that pass through the same path/road of the input trajectory. However, existing trajectory similarity search [12] or trajectory join [11] implementations may consider the input trajectory to be very different from any underlying trajectory because of their different resolutions and lengths. As a result, we need to revisit new implementations of trajectory exploration query. In the following, we present the formal definition of the trajectory exploration query.

Following previous works, such as [11], we define a trajectory as follows. Note that, TraV is not restricted by the point-based representation used in this section and can be adapted to use other representations such as segment-based trajectory data with ease.

Definition 2.1. A **trajectory** of a specific object is an ordered sequence of points, denoted by $T =< p_1, p_2, ..., p_n >$, where each p_i is a coordinate corresponding to the same object. Further, these coordinates are ordered based on their timestamp, such that $T(p_i) < T(p_{i+1})$, where $T(p_i)$ represents the timestamp associated with p_i.

Definition 2.2. Given two trajectories, $T_l =< l_1, l_2, ..., l_n >$ and $T_r =< r_1, r_2, ..., r_m >$, if there exist two points r_i and r_j in T_r such that $i < j$ and both $dist(r_i, l_1)$, $dist(r_j, l_n)$ are smaller than a threshold δ. Then, all consecutive points between r_i and r_j in T_r form a **candidate sub-trajectory** that needs to be explored corresponding to T_l.

Note, that there might be multiple sub-trajectories that could be constructed from T_r that satisfy the above conditions and all such trajectories need to be explored by the system to ensure high accuracy.

Definition 2.3. Given two trajectories, T_l, T_r, T_r is a **near-by** trajectory of T_l if there is at least one candidate sub-trajectory, T'_r, of T_r such that $sim(T'_r, T_l)$ is smaller than a predefined threshold. Finally, $sim(x, y)$ can be computed using any similarity metric such as Fréchet distance [8], dynamic time warping [9] etc.

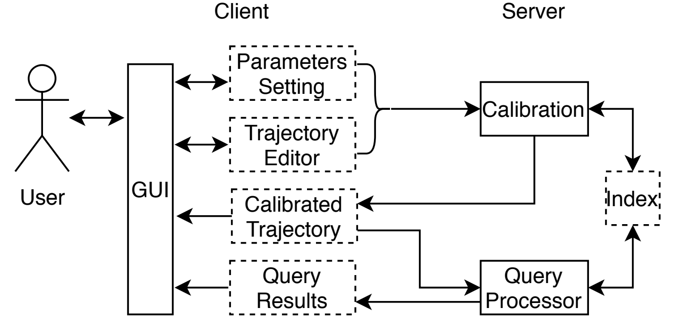

Fig. 1: Architecture of TraV. Solid rectangles represent major components and dotted rectangles represent auxiliary data structures/parameters.

Finally, we define trajectory exploration query as follows.

Definition 2.4. Given an input trajectory T_q=<$p_1, p_2, ..., p_n$>, and a set of N trajectories (<$T_1, T_2, ..., T_N$>), the **trajectory exploration query** should identify all trajectories ($< T_1, T_2, ..., T_k >$) that are near-by trajectories of T.

3. SYSTEM ARCHITECTURE

The system architecture of TraV is shown in Figure 1. TraV adopts a client-server model to allow wide usage of the system by a variety of users while also taking advantage of the massive computational capability of modern server hardware. Overall TraV comprises of three main units: 1) a GUI unit that allows users to express their interested trajectory by simply drawing on a map of a geographical region using a mouse or other input means (touch, stylus etc.), 2) a Calibration Unit that calibrates the input trajectories being expressed by the user based on the underlying road network data and 3) a Query Processing Unit that is responsible for exploring the underlying trajectory data corresponding to the user input.

3.1. Graphical User Interface

The Graphical User Interface (GUI) of TraV runs on the client side and allows the user to express his/her trajectory. Furthermore, it is connected to both the Calibration Unit and the Query Processing Unit which are both running on the server in a remote location.

Initially, the user is presented with an overview of the map of a geographical region. Note, the user can freely interact with the map (e.g move to a different geographical region, zoom in, zoom out etc). As soon as the user starts drawing, the GUI starts sampling the user mouse/touch input and sends them to the Calibration Unit running in the server, as two dimensional coordinates (Latitude, Longitude). Overall, the GUI does not wait for the entire input trajectory to be expressed and progressively sends the input data to the server for processing to ensure very fast response to the user query.

As the user is expressing his/her trajectory, the calibrated results will be presented back to the user. Furthermore, the user can go into an "Edit" mode to approve or ignore the calibrated results. This is especially useful when trying to find missing roads or road segments in the underlying road network. Once the user determines the appropriate calibration points, the calibrated trajectory is transmitted back to the server. The server then returns the nearby trajectories associated with the calibrated input trajectory, which is presented to the user. Note, a video demonstrating the use of our GUI can be found in the supplementary materials.

3.2. Calibration Unit

TraV allows users to simply draw on a GUI to express their interested trajectory. However, hand drawing is hardly accurate and calibration of the user input is essential to fix any mistakes in user input and to provide a good user experience. To this end, our Calibration Unit maps each input coordinate from the user to a corresponding GPS coordinate in the underlying road network data obtained from the Land Transport Authority [1]. Note, the major challenge in this calibration stage is choosing a route that is geographically close to the coordinates expressed by the user, while also taking into consideration the likelihood of choosing a given route. This is especially important in regions like Singapore which has extremely dense road networks. To achieve this, we adopt a Hidden-Markov-Model (HMM) based map matching algorithm presented in previous studies [10, 7] that takes into consideration both the likelihood of taking a specific route (based on the previous trajectory points already expressed by user) and the geographical distance between a point expressed by the user and a route in the underlying road network.

Conventional implementations often adopt a "query-before-process" execution model. Such an execution model introduces unnecessary latency, which is undesirable in an interactive trajectory exploration system. Hence, we implement the Calibration Unit in TraV as a streaming application running on Flink [2], which is capable of performing the calibration as the user is expressing his/her input trajectory.

Algorithm Overview. Overall, our calibration algorithm is implemented as follows.

- As mentioned before, the input coordinates are continuously streamed into the Calibration Unit by the GUI as the trajectory is being expressed by the user. The Calibration Unit first performs down-sampling on this input data. This is because directly treating the points sampled by the GUI is both computationally intensive and unnecessary, especially if the underlying trajectory data has a significantly lower resolution.
- Next, the Calibration Unit identifies potential GPS coordinates in the underlying road network, for each input coordinate. Naively, all points in the underlying road network data are candidates for this computation. To minimize the computation, our Calibration Unit makes use of a simple R-Tree based indexing to first identify a set of road network points which are within δ meters of each input coordinate. Note, δ is a tune-able parameter, which can be varied by users.
- Finally, the Calibration Unit applies the Viterbi algorithm [6] to compute the most likely sequence of calibrated GPS coordinates as the calibrated trajectory.

Algorithm Implementation. We implement the above calibration algorithm as a continuous streaming application on Apache Flink [2], consisting of the following operators: 1) a `Spout` that continuously receives input coordinates from the GUI, 2) a `Localization` operator that identifies the candidate GPS coordinates in the road network corresponding to each input coordinate (using an R-Tree Index and a predefined threshold), 3) a `PreCalibration` operator that computes the emission probabilities or the likelihood of each candidate point identified by the Localization operator based on their geographical distance from the point plotted by the user, 4) a `PostCalibration` operator that computes the transition probabilites or the likelihood of the user moving to each one of the candidate points from the previous point and 5) a `Sink` that sends the point with the highest probability back to the visual interface so that it can be presented back to the user. All operators except the PostCalibration operator can be parallelised in our implementation.

3.3. Query Processing Unit

The Query Processing Unit in TraV is responsible for identifying the nearby trajectories of the calibrated input trajectory provided by the user (or non calibrated if all calibration points are ignored by the user). However, the process of identifying nearby trajectories from billions of underlying trajectory points is computationally intensive and could hence lead to a poor user experience in an interactive system like TraV.

To minimize the amount of computation and for quick processing of large amounts of trajectory data, TraV adopts the following techniques. First, to minimize the number of candidate sub-trajectories that are explored, we build an R-Tree index on the entire underlying trajectory data set. The Query Processing Unit module then only selects those sub-trajectories that begin and end within a specified threshold of the start and end points of the calibrated trajectory for exploration. Second, we leverage Flink and its parallel processing engine to make efficient use of modern massively parallel multi-/many- core hardware to quickly compute the distance between each candidate sub-trajectory and the calibrated input trajectory.

Algorithm Overview. Overall our Query Processing Unit works as follows.

- First, the Query Processing Unit receives the (calibrated) input trajectory from the visual interface.

- It then quickly retrieves a set of candidate sub-trajectories which are within a threshold distance of the start and end points of the calibrated input trajectory. The candidate sub-trajectories are then grouped together by their parent trajectory ID to leverage overlapping of GPS coordinate data across sub-trajectories.
- The distance between each group of sub-trajectories and the calibrated input trajectory is then computed by a predefined number of threads in parallel. To further minimize the cost of computing the distance of each group of sub-trajectories with the input trajectory, we leverage the overlap in GPS coordinates across different sub-trajectories of the same parent trajectory.
- Finally, the trajectories which have at least one sub-trajectory within a threshold distance of the calibrated input trajectory is identified as nearby trajectories.

Note, in this study we use the Fréchet metric [4] for measuring the distance between a sub-trajectory and the calibrated input trajectory. Our implementation can be easily modified to use any other metrics [5] like Edit Distance, Synchronized Euclidean Distance, Dynamic time warping etc.

Algorithm Implementation. The above algorithm is implemented as a streaming application using Apache Flink and consists of the following operators: 1) a `Spout` that receives the calibrated input trajectory from the GUI, 2) a `CandidateGenerator` which generates the candidate sub-trajectories which are within a threshold distance of the start and end points of the calibrated input trajectory, 3) a `DistanceCalculator` that computes the distance between each group of sub-trajectories and the calibrated input trajectory and 4) `Sink` that sends the trajectories within a threshold distance of the calibrated input trajectory to the GUI to be presented to the user. All operators except the `CandidateGenerator` in our implementation can be executed using multiple threads. The `DistanceCalculator` operator is the most computationally intensive one, and is hence allocated relatively more computing resources.

4. DEMONSTRATION

4.1. Demo Setup

Our demonstration setup consists of a client and a server. The client device (Mobile or Laptop PC) runs the GUI module depicted in Figure 1 and the server (a workstation located in our Lab at NUS, Singapore) runs the calibration and query processing module. The user can then draw the trajectory on the GUI module as described in Section 3.

Our demonstration uses trajectory data collected from Grab drivers in Singapore. The underlying trajectory data set consists of billions of GPS coordinates and our massively parallel trajectory exploration application is able to quickly satisfy most user queries and ensure an interactive user experience. Further, we use road network data collected from Land Transport Authority [1] consisting of 250K GPS coordinates.

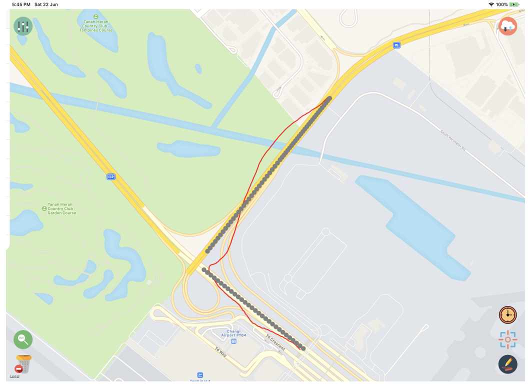

Fig. 2: Auto Calibration in TraV.

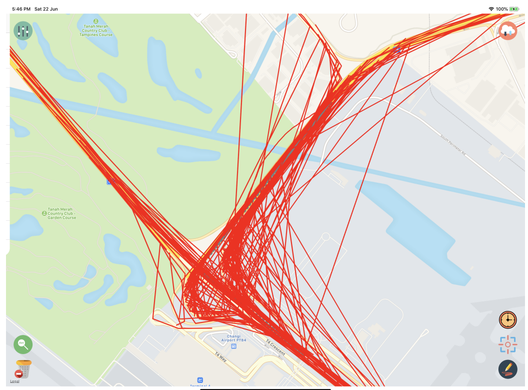

Fig. 3: Result of trajectory exploration in TraV (explored trajectories shown in red).

4.2. Demonstration Objectives

The objectives of our demonstration are: 1) demonstrate the effectiveness of our automatic calibration in assisting the user in expressing the trajectories. 2) demonstrate the ability of our system to quickly explore large trajectory data sets.

The demonstration workflow is as follows. First, we allow the user to draw trajectories using our GUI module and the GUI will present the calibrated results while the trajectory is being populated by the user (Figure 2). Second, once the user makes a determination of the appropriate trajectory data for exploration (i.e either the original input trajectory or the calibrated trajectory), TraV starts processing and presents the nearby trajectories back to the user along with the time taken for the computation, helping the user to understand the capabilities of our machine (Figure 3) .

5. ACKNOWLEDGEMENT

This work was funded by the Grab-NUS AI Lab, a joint collaboration between GrabTaxi Holdings Pte. Ltd. and National University of Singapore. We would like to thank Prof. See Kiong Ng, Xiang Hui Nicholas Lim and Yong Liang Goh for their support.

6. REFERENCES

[1] Land transport authority. https://www.mytransport.sg/content/mytransport/home/dataMall.html.

[2] Apache flink. https://flink.apache.org/, 2018.

[3] Grab – transport, food delivery & payment solutions. https://www.grab.com/, 2019.

[4] H. Alt and M. Godau. Computing the fréchet distance between two polygonal curves. *International Journal of Computational Geometry & Applications*, 5(01n02):75–91, 1995.

[5] L. Chen, M. T. Özsu, and V. Oria. Robust and fast similarity search for moving object trajectories. In *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, SIGMOD '05, pages 491–502, New York, NY, USA, 2005. ACM.

[6] G. D. Forney. The viterbi algorithm. *Proceedings of the IEEE*, 61(3):268–278, 1973.

[7] C. Y. Goh, J. Dauwels, N. Mitrovic, M. T. Asif, A. Oran, and P. Jaillet. Online map-matching based on hidden markov model for real-time traffic sensing applications. In *2012 15th International IEEE Conference on Intelligent Transportation Systems*, pages 776–781, Sep. 2012.

[8] S. Har-Peled and B. Raichel. The fréchet distance revisited and extended. In *Proceedings of the twenty-seventh annual symposium on Computational geometry*, pages 448–457. ACM, 2011.

[9] M. Müller. Dynamic time warping. *Information retrieval for music and motion*, pages 69–84, 2007.

[10] P. Newson and J. Krumm. Hidden markov map matching through noise and sparseness. In *Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, GIS '09, pages 336–343, New York, NY, USA, 2009. ACM.

[11] S. Shang, L. Chen, Z. Wei, C. S. Jensen, K. Zheng, and P. Kalnis. Trajectory similarity join in spatial networks. *Proc. VLDB Endow.*, 10(11):1178–1189, Aug. 2017.

[12] D. Xie, F. Li, and J. M. Phillips. Distributed trajectory similarity search. *Proc. VLDB Endow.*, 10(11):1478–1489, Aug. 2017.