A Web-Based System for Efficient Contact Tracing Query in a Large Spatio-Temporal Database

Shadman Saqib Eusuf
BUET, Dhaka
s.saqibeusuf@gmail.com

Kazi Ashik Islam
University of Virginia
ikaziashik@gmail.com

Mohammed Eunus Ali
BUET, Dhaka
eunus@cse.buet.ac.bd

Sifat Muhammad Abdullah
BUET, Dhaka
sifat.abdullah577@gmail.com

Abdus Salam Azad
University of California, Berkeley
azadsalam@cs.berkeley.edu

ABSTRACT

In this demonstration, we present a web based system for the novel contact tracing query (CTQ) that finds users who have come into *direct contact with the query user* or *indirect contact via the already contacted users* from a large spatio-temporal database. The CTQ is of paramount importance in the era of new COVID-19 pandemic world for identifying people who came into close spatial and temporal proximity with persons carrying an infectious disease. We demonstrate a multi-level index named QzR-tree, that considers the space coverage and the co-visiting patterns of the trajectories to group users who are likely to meet. More specifically, we use a quadtree to partition user movement traces along with a linear ordering and use the space-time mapping to group users with an R-tree. We develop a web-based demo system to show the effectiveness of the QzR-tree for the CTQ. The web-based system essentially uses a PostgreSQL database to store user trajectories, and indexes these trajectories using the QzR-tree, and finally uses a web interface to take user query and display the results in a map.

CCS CONCEPTS

• **Information systems** → **Spatial-temporal systems**; *Search interfaces.*

KEYWORDS

Contact tracing query system; Trajectory database

ACM Reference Format:
Shadman Saqib Eusuf, Kazi Ashik Islam, Mohammed Eunus Ali, Sifat Muhammad Abdullah, and Abdus Salam Azad. 2020. A Web-Based System for Efficient Contact Tracing Query in a Large Spatio-Temporal Database. In *28th International Conference on Advances in Geographic Information Systems (SIGSPATIAL '20), November 3–6, 2020, Seattle, WA, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3397536.3422350

SIGSPATIAL '20, November 3–6, 2020, Seattle, WA, USA

ACM ISBN 978-1-4503-8019-5/20/11.
https://doi.org/10.1145/3397536.3422350

1 INTRODUCTION

In this demonstration, we present a web-based system to illustrate the contact tracing query (CTQ) in a spatio-temporal database. Suppose D is the historical mobility traces (i.e. trajectories) of users obtained from GPS-enabled phones or mobile signals through triangulations over the last T days. Each user $u \in D$ is represented as a sequence of timestamped locations $\{(l_1, t_1), (l_2, t_2)...(l_n, t_n)\}$ where her visited places are l_1, l_2, ..., l_n at different times t_1, t_2, ..., t_n, respectively. Let us assume that q be the trajectory of a patient having an infectious disease (e.g. COVID-19). The CTQ identifies a set of users $U \subset D$ who have come into *direct contact* with q, and subsequently finds those who have come into *contact* with the already contacted users.

Processing different trajectory related queries such as range, join, nearest-neighbor, etc. has been addressed in the literature [1, 4, 5, 7] with several trajectory indexing schemes. These are variants of traditional spatio-temporal indexes like *R*-tree [3] or quad-tree [6] and tailored for answering those queries efficiently. The CTQ cannot be solved using these existing indexes by running repetitive range queries for the points of the query trajectory as that would make it extremely in-efficient. This is because (i) the mobility traces of a user usually span a set of dispersed points with timestamps covering a large area, different from the normal point data such as POIs (Point of Interest) or trajectory data such as taxi trips, and thus traditional indexes cannot prune them efficiently, (ii) if a user's historic trajectory matches with the query at any point, spatially and temporally, then she will be a candidate answer, and may infected others subsequently, which requires running the query recursively.

We propose a two-level index structure, namely QzR-tree, to answer the CTQ efficiently. It exploits the strengths of both quad-tree and R-tree. In the first level, we use a quad-tree to partition the points of historical trajectories, where the location of a point is specified by the smallest quad-tree block containing it. Similarly, the timestamp of each location of a trajectory is mapped to a time bucket entry. Then we transform the trajectories to a new coordinate system for the trajectory points. Next, we apply an R-tree on the trajectory points in the new coordinate system to organize the trajectories with co-visiting pattern in disk. We present a divide-and-conquer approach to answer CTQ efficiently, which runs through different levels of the index dividing the query trajectory recursively.

We develop an interactive web-based system that demonstrates all the above steps of processing CTQ. We use PostgreSQL, an open

source database, to host our huge trajectory database at the back-end. Then we build our index and query processing techniques in the middleware. Finally, as a front-end, we build a web-based interactive system that uses a map-based visualization for taking inputs from a user and displaying the output results of CTQ.

The rest of the paper is organized as follows. We formally define the CTQ in Section 2. Next, we discuss our proposed index in Section 3 followed by the query processing algorithm and evaluation in Section 4. Then we provide the details of our demonstration system in Section 5 and conclude the paper in Section 6.

2 PROBLEM FORMULATION

Let D be the trajectory dataset, where each user $u \in D$ is represented as a sequence of timestamped locations $\{(l_1, t_1), (l_2, t_2), \ldots (l_n, t_n)\}$ denoting her visited places $l_1, l_2, ..., l_n$ at different times $t_1, t_2, ..., t_n$, respectively. Let q be the query user carrying an infectious disease (e.g., COVID-19).

Condition of u meets v: Let $\{(u.l_1, u.t_1), (u.l_2, u.t_2)...(u.l_n, u.t_n)\}$ and $\{(v.l_1, v.t_1), (v.l_2, v.t_2)...(v.l_m, v.t_m)\}$ be two sequence of timestamped locations of u and v, respectively. Let *spatialDist* and *temporalDist* be the spatial distance and temporal distance measuring functions between two locations and two timestamps. Now, for any $i \in [1, n], j \in [1, m]$, if *spatialDist* $(u.l_i, v.l_j) \leq \psi$ and *temporalDist* $(u.t_i, v.t_j) \leq \tau$, then we say the trajectory u *meets* the trajectory v. Here, ψ and τ, are spatial (euclidean) and temporal distance thresholds. Note that, in order for the disease to transmit, it is required but not sufficient that two trajectories *meet*.

The objective of the CTQ is to identify the set of users $U \subset D$ where each user $u \in U$ is at risk of being infected for being in close spatio-temporal proximity with users potentially carrying the infectious disease. We define the set U as follows.

(1) Let U_0 be the set of users where each user $u \in U_0$ *met* q at any timestamp $u.t$ in the last T days. We say that u was at risk from time $u.t_{risk}$ $(= u.t)$. Here, the subscript (zero) in U_0 denotes that, the number of intermediate carriers (of the virus) between user q and u is zero (i.e u was infected by the query user q).
(2) Let U_i be the set of users where each user $u \in U_i$ *met* with any user $v \in U_{i-1}$ at timestamp $u.t > v.t_{risk}$. Now we can define the set U as union over all U_i's upto a certain level, which works as the maximum recursion depth parameter.

Next, we define our contact tracing query as follows.

Definition 2.1. (CTQ). Given a set D of user trajectories, an infected user trajectory q, a spatial proximity threshold ψ, a temporal proximity threshold τ, and an integer L, a CTQ query finds a set of users $U \subset D$ such that $U = \bigcup_{i=0}^{L-1} U_i$; where U_i is the set of users who are at risk of being infected via 'i' number of intermediate carrier users.

3 THE PROPOSED INDEX

The trajectories in our datasets can be very long (e.g., 14 days of mobility traces) and may cover large areas. So, using an R-tree to index them might not be beneficial as their MBRs (Minimum Bounding Rectangle) will most likely overlap too much.

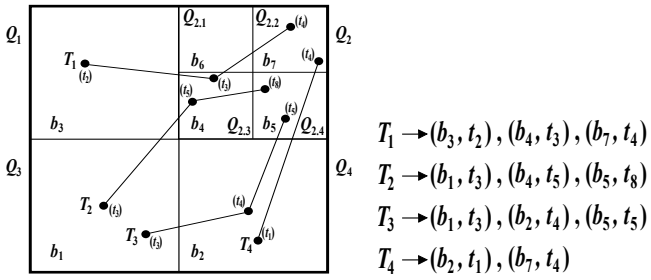

Figure 1: (a) A quadtree based space partitioning of trajectories. (b) Mapping of trajectories.

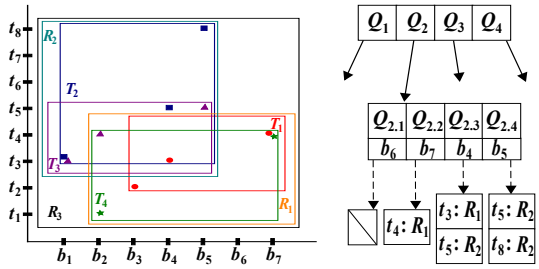

Figure 2: (a) R-tree based grouping of trajectories in a transformed space. (b) QzR-tree index structure.

3.1 QzR-tree

The key intuition of our proposed index is, the trajectories whose points are co-located at the overlapping time-instant are likely to match with the same query. Based on this observation, we present a two-level index, the Quad R tree with z-ordering (QzR-tree) combining the benefits of both quadtree and R-tree.

Firstly, the spatial region is recursively partitioned using quadtree to accommodate the trajectory points. A leaf of the quadtree contains at most θ points. Then we order the quadtree blocks with a space filling curve, specifically a z-curve (Morton order) and assign each of them a number. Let us call this number the $spatial - id$ of the block. Thus, in the spatial domain each trajectory is represented as a list of $spatial - id$s. Similarly, each timestamp of a trajectory is mapped to a time bucket and assigned a number, $temporal - id$. Thereby, each trajectory can now be represented as sequence of $(spatial - id, temporal - id)$ tuples. This mapping of the trajectories can be seen as a transformation to a new coordinate system, where x and y-axis represents spatial and temporal dimensions respectively.

In the next step, we use an R-tree to group trajectories based on their points in the newly transformed coordinates. Each set of points of a trajectory is represented as an MBR, and the R-tree groups close-by MBRs in a leaf node which is stored in a disk-page. We save this *disk-page id* in all the quadtree leaf-blocks that contain points of the trajectories stored in this disk-page. We also maintain associated temporal ids in the leaf-blocks, denoting the time range of trajectory points stored in the corresponding disk-page. Note that, we only use the R-tree to group co-visiting trajectories, computed from the transformed space, rather than keeping its hierarchical structure.

Fig. 1 and Fig. 2 show the construction process of the QzR-tree. Figure 1(a) shows an example with four user trajectories, $\{u_1, \ldots, u_4\}$, where $\theta = 2$. The space is first divided into four quadrants $Q_1, \ldots, Q_4$. As Q_2 contains more than 2 trajectory points, this block is further divided into $Q_{2.1}, \ldots, Q_{2.4}$. We then apply z-ordering to number these quadtree blocks as $b_1, b_2, ..., b_7$. Besides the timestamp of each point in the trajectories is assigned time-bucket number between t_1 and t_8. Fig. 1(b) shows the new representation of trajectories as a sequence of (b_i, t_j) tuples. The points are then mapped into a new coordinate system in a two-dimensional space (Fig. 2(a)), where we can see the four sets of points of the trajectories in four different colors, each enclosed in an MBR. These MBRs are grouped together to form an R-tree. Each leaf level node, R_1, R_2 etc. corresponds to a disk-page. Finally, we maintain these disk-page references in different quadtree leaf-blocks of the QzR-tree, as shown in (Fig. 2(b)). For example, with $Q_{2.4}(b_5)$, disk-page id R_2 is assigned along with time-bucket ids t_5 and t_8.

4 PROCESSING CTQ AND EVALUATIONS

Algorithm: We develop a divide-and-conquer based best-first algorithm for processing CTQ using our proposed QzR-tree. Intuitively, the quadtree of QzR-tree supports faster range query around query trajectory points, while R-tree grouping helps reduce I/O overhead. We apply a spatial pruning followed by a temporal pruning using QzR-tree. The irrelevant quadtree nodes are pruned first, and then the time buckets are used to further prune the R-tree blocks to be retrieved. Intuitively, we hierarchically traverse tree by progressively pruning different branches. So we look up our in memory QzR-tree index and obtain a list of relevant $R-tree$ nodes (i.e. disk-page ids). We then fetch the trajectories (T_r) stored in those disk blocks. For each user trajectory $t_i \in T_r$, we compute whether the user *meets* with q and include her in the set U accordingly. Note that, these users are at risk of being infected because of being spatio-temporally co-located directly with a person having an infectious disease. We recursively process the CTQ with each of the newly found users at risk, as query users, until maximum recursion depth is reached, and find the users trajectories at risk of being infected indirectly via intermediate carrier users.

Evaluation: We evaluate the QzR-tree with a baseline (**BL**) approach using a 3D R-tree. We use the mobility traces from the CDR data collected by Grameenphone Ltd, referred as *BD Cellphone*, and Foursquare check-in dataset of New York city to evaluate performance of CTQ. The experimental result show that QzR-treetree outperforms the BL by 1-2 orders both in terms of processing time and I/O.

The details of the index, algorithms, and experimental results can be found in our technical report [2].

5 DEMONSTRATION

Our web-based demonstration system[1] consists of the following three major components: 1) back-end PostgreSQL database; 2) indexing and query processor; 3) visualization. We briefly describe each component below.

Back-end PostgreSQL Database: The *BD Cellphone* dataset consists of about 4.5 million trajectories having around 110 millions timestamped locations in total. We use PostgreSQL database to store this large number of mobility traces. First, we load the data to a table where each row corresponds to a single trajectory point and has the anonymous trajectory id, latitude, longitude and timestamp as the fields (i.e. columns). Then, we group together the points that belong to the same trajectory and write the trajectories to a separate table. Each row in this *trajectory* table corresponds to a single trajectory. Finally, we cluster (i.e physically reorder) the data of the *trajectory* table in disk, according to our proposed QzR-tree index for efficient retrieval during CTQ processing.

Indexing and Query Processor: This is the middleware component of the system and is implemented using Java (JDK 1.8). To efficiently process the CTQ, we build a QzR-tree index for accessing the trajectories. On top of the index, we implement our best-first strategy to process the CTQ. This middleware takes a user trajectory and other parameters as inputs, processes the query by fetching the required trajectories from the database, computes the result and returns it to the visualizer. During the index construction, we set θ = 128 points, γ = 30 minutes and β = 4 trajectories, as these best fit with the data properties.

Visualization: The user interface of the system is a web page created using dynamic HTML, and Python with Django (the framework). It has four main sub-components: (1) Settings Panel for setting up the query parameters, (2) Input Panel for selecting a user (or a group of users) trajectory as an input, (3) Visualization Panel to view user traces and contact traces on a map, and (4) Ranking Panel to rank results based on the infection likelihood.

Settings Panel. This is a pop-up left panel of the user interface that includes options to set different query parameters, (i) spatial range threshold, (ii) temporal range threshold, (iii) level (maximum recursion depth) of the contact trace, and (iv) the ranking function. The user may choose different range and values by using sliders or check-boxes. These query parameters are passed as along with the query trajectory. We allow spatial range between 1-5 meters, temporal range between 1-10 minutes, the depth level as 1 and 2, and ranking function based on the spatial or temporal order. All these parameters can be set to support various infectious disease (e.g., COVID-19) characteristics.

Input Panel. This is the right panel of the display, where a user can give a query user id, for whom we want to trace contacts, existing in the database. Some example ids and their trajectories are summarized in a link given just right after the input box. A user can also choose to upload a list of query user ids to do the combined contact tracing.

Visualization Panel. The main part of the visualization is shown in the Open-Street Map. It has three states: (i) mobility trace of a query, (ii) contact traces of the query, and (iii) heat-map of the infected persons in the database. When a user gives an id to the input panel and press the 'Mobility Trace' button, the entire trajectory of the corresponding person is shown on the map with a provision of zooming to different spatial granularities. On the other hand, when the user presses the 'Contact Trace', the system runs the CTQ for the given query id, and display the traces of contacted persons on the map. The user can see the contact traces people who are at risk (for the chances of being infected directly by a patient or indirectly by intermediate users), on the map along with the

[1] http://ctracing.datalab.buet.io/

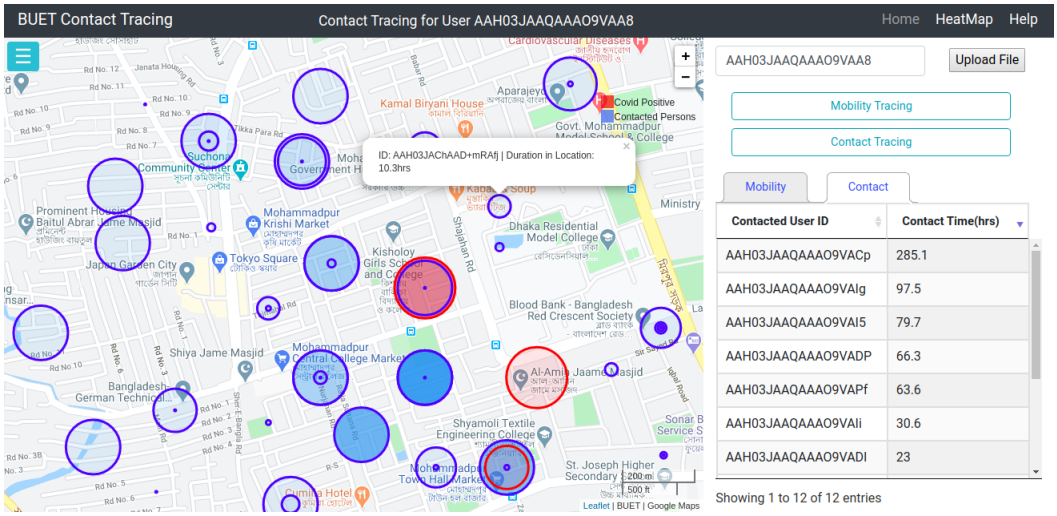

Figure 3: The Contact Tracing System

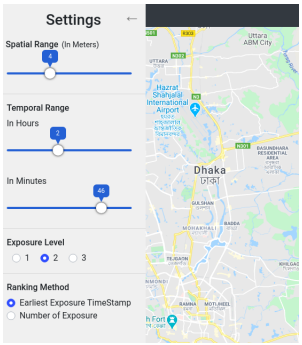

Figure 4: Setting Parameters

mobility traces of the query user. As a last item, a user can see the heat map of already infected patients by pressing the 'Heat Map' link from the top menu, which essentially shows the hotspots based on their spatial density.

Ranking Panel. This panel is on the right shows the results based on two ranking criteria. The first one is to rank the users by the temporal order, in which the person who has been at risk for coming into contact at the earliest will be ranked the highest. The second criterion is to rank the users by the number of times they have been come into contact with a patient or an intermediate carrier.

Demonstrated Scenarios. Fig. 3 shows a screenshot of the contact tracing system. (i) The left map panel shows the contact tracing for query user id 'AAH03JAAQAAAO9VAA8'. The mobility trace of the query user is shown in red circles, and the contacted users are shown as blue circles. The circle radius is related to time spent at that location (longer time shows bigger circle), and, the opacity is determined by frequency of visit (darker shades for more visits). (ii) The right panel for giving user query id as input (or uploading a cvs file with a group of query ids). (iii) A two-tab list, where the highlighted 'Contact' tab shows the list of contacted users in order of their *meeting* time with a disease or intermediate carrier, configurable from the settings panel. The other 'Mobility' tab shows the ranks of different regions based on the frequency of visits by the given query user. Fig. 4 shows the parameter settings panel, where the user can set different query parameters. Other screenshots are omitted for brevity and the reader is encouraged to try the live system at http://ctracing.datalab.buet.io/.

6 CONCLUSIONS

We have demonstrated a web-based system for the novel CTQ in the context of spatio-temporal databases. We build our multi-level index, namely QzR-tree, as middleware of the system to efficiently process the CTQ. We use PostgreSQL as a back-end database to store trajectories, and Python with Django (the framework) in the interactive front-end to visualize the results in a map. We argue that without the loss of generality our proposed system can be used for identifying a potential set of people who have been in contact with the real infected person from trajectories. While the exact spatial and temporal range of the contacts may vary from disease to disease , our proposed technique is still applicable. Hence, the CTQ system will be an immensely valuable tool for large-scale health surveillance in a pandemic scenario.

REFERENCES

[1] Mohammed Eunus Ali, Shadman Saqib Eusuf, Kaysar Abdullah, Farhana Murtaza Choudhury, J. Shane Culpepper, and Timos Sellis. 2018. The Maximum Trajectory Coverage Query in Spatial Databases. *PVLDB* 12, 3 (2018), 197–209.

[2] Mohammed Eunus Ali, Shadman Saqib Eusuf, and Kazi Ashik Islam. 2020. An Efficient Index for Contact Tracing Query in a Large Spatio-Temporal Database. https://arxiv.org/pdf/2006.12812.pdf/. arXiv:2006.12812 [cs.DB]

[3] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. 1990. The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles. In *SIGMOD*. 322–331.

[4] V Prasad Chakka, Adam Everspaugh, Jignesh M Patel, et al. 2003. Indexing large trajectory data sets with SETI.. In *CIDR*, Vol. 75. 76.

[5] Mohamed F. Mokbel and Walid G. Aref. 2009. *Indexing Historical Spatio-Temporal Data.* 1448–1451.

[6] Hanan Samet. 1984. The Quadtree and Related Hierarchical Data Structures. *ACM Comput. Surv.* 16, 2 (1984), 187–260.

[7] Shuo Shang, Lisi Chen, Zhewei Wei, Christian S. Jensen, Kai Zheng, and Panos Kalnis. 2017. Trajectory Similarity Join in Spatial Networks. *PVLDB* 10, 11 (2017), 1178–1189.