Travel Time Estimation in Real-Time using Buses as Speed Probes

Dimitrios Tomaras, Ioannis Boutsis, Vana Kalogeraki
Department of Informatics
Athens University of Economics and Business
Email: {tomaras,mpoutsis,vana}@aueb.gr

***Abstract*—Travel time estimation is a strategically important service in urban environments for personalized and eco-friendly route planning optimization, congestion avoidance, ridesharing and taxi dispatching. However, storing and retrieving traffic data in specific spatiotemporal regions is not an easy task as the data generated by these systems are typically very large and dynamic. In this paper we propose an efficient and scalable solution for real-time travel time estimation of trajectories. In our system buses are used as speed probes to obtain real-time traffic data information and spatio-temporal trajectories are stored in a dynamic indexing system optimized for efficiently retrieving spatiotemporal data in real-time. Our experimental evaluation illustrates the efficiency and scalability of our approach.**

1. Introduction

Recently we have observed the proliferation of "location-based systems" as a consequence of the ubiquitousness of GPS sensors in location-aware smartphone devices, car navigation systems, etc. Such devices have made it possible to monitor and track the movement of pedestrians, cyclists, motorists and humans in public transport in real-time and has enabled us to identify interesting events.

The availability of such information has led towards developing a wide range of applications which are characterized as "location based services". Such services provide user-centric information that may include estimating traffic conditions, locating nearby stores, local weather, etc. For instance, in EcoTour [3] the authors exploit GPS and historical data for proposing environmental friendly routes, while in StartTrack Next Generation [7] they exploit series of GPS traces to support ridesharing and personalized driving direction services.

A fundamental challenge in such services is that they rely on the use, storage and real-time analysis of tremendous amounts of data. Such data typically describe user movement patterns or other spatiotemporal instances that are dynamically generated from multiple types of participating sensors, and the queries typically require real-time information and fast responses. Thus, maintaining and processing queries on data that change rapidly over time is a difficult task.

Several schemes have been proposed that focus on maintaining geo-located information efficiently, most of which depend on tree structures. One of the most popular methods, R-trees [6], aims to assign each geo-located point in the respective spatial container but they do not consider the time dimension. Other approaches, such as STR-trees [2] consider spatiotemporal instances but they do not take into account the sequence of the points produced by the same sensor, that reflect its trajectory. Finally, Trajectory Bundle Trees (TB-Trees) [9] take into account the sequence of spatiotemporal instances, but they are not optimized for real-time spatiotemporal queries.

In this paper we propose an efficient and scalable solution for real-time travel time estimation of trajectories, using buses as speed probes. We develop TRavEl Estimation Trajectory (sTREET), a system optimized for efficiently estimating travel time by considering spatiotemporal information retrieved from buses. sTREET indexes lines of buses using the line they travel as a key. It stores information about the current road network conditions. By keeping a secondary index, sTREET manages to optimize the time needed for processing spatiotemporal queries.

Our main contributions are summarized as follows:

- We develop sTREET a novel tree-based structure that is designed to index large and dynamic spatiotemporal trajectories.
- sTREET is optimized for real-time spatiotemporal queries and can be used to compute travel times by exploiting real-time information produced from buses.
- sTREET takes advantage of a secondary index that significantly reduces the search space for our defined spatiotemporal queries by an order of magnitude.
- Our experimental evaluation shows that sTREET is practical, efficient and can effectively retrieve spatiotemporal data produced in real-time to meet application demands.

2. System Model and Overview

In this section we present our system model and we introduce our tree structure that handles spatiotemporal trajectories.

2.1. Smart City Application

Smart cities use digital technology devices embedded across all city functions, to provide important services for the citizens and businesses on several aspects including traffic management, environment monitoring, housekeeping information, etc. Dublin is a smart city [8] that that takes advantage of such devices for managing traffic congestions and incidents. Traffic management is controlled by the Dublin City Council, a local authority that develops, maintains and manages the city's road network for the benefit of pedestrians, cyclists, motorists and public service and commercial vehicles.

In Dublin, the main mode of public transport, in terms of passenger numbers and coverage, is by bus. The main bus operator has a fleet of over a thousand buses, most of which pass through the city center. For instance, at Trinity College there are almost 400 buses during the rush hour (08:00-09:00) in both directions. Since 2008, these buses are fitted with an Automatic Vehicle Location (AVL) system that provides real-time GPS traces; these are collected with a 30-second interval and are available in real-time. Such traces can provide significant information to estimate traffic congestion and calculate alternative routes. However, processing such data in real-time for all interested commuters is a challenging process due to the size of the data produced. In this work we focus on optimizing such queries so as to execute them efficiently and in real-time.

2.2. System Model

Definition 1 **Traffic Monitoring System.** We consider a traffic monitoring system where buses are equipped with Automatic Vehicle Location systems that are used to generate real-time traffic data reporting GPS location, direction and speed. During a trip, the bus periodically reports its current position, destination and speed; these information are then utilized to compute road conditions as well as speed limits on the streets. Each bus is considered as a moving sensor i that generates j records r_{ij} of the following form: r_{ij}:$\langle lat_{ij}$, lon_{ij}, $time_{ij}$, $direction_{ij}$, $lineId_{ij}$, $vehicleJourneyId_{ij}$, $vehicleId_{ij}$, $delay_{ij}\rangle$, where lat_{ik}, lon_{ik} define the spatial location of the bus, in terms of latitude and longitude, $time_{ij}$ represents the unix timestamp when the record was produced, $direction_{ij}$ shows the geographical direction of the bus, $lineId_{ij}$ is the identifier of the line, $vehicleJourneyId_{ij}$ defines the identifier of the pattern of the journey, and $vehicleId_{ij}$ denotes the identifier of the specific bus.

Definition 2 **Trajectory.** A bus trajectory $traj_k$ is a sequence of spatiotemporal time-ordered records r_{ij} with the same $lineId_{ij}$, $vehicleId_{ij}$ and $vehicleJourneyId_{ij}$.

Definition 3 **Temporal Estimation.** For each bus route, we assume a spatial decomposition of the route into non-overlapping segments seg_e and we estimate the time needed to reach every segment seg_e, denoted as t_e. Thus, for a given route we extract all trajectories $traj_k$ with the same $lineId_{ij}$ and $vehicleJourneyId_{ij}$ and we add them to the list $Route$. For each record $r_{ij} \in traj_k, \forall traj_k \in Route$ we subtract the value of $time_{ij}$ with the time point when the trajectory of r_{ij} has started: $time_{ij} = time_{ij} - \min(time_{mn} \forall r_{mn} \in traj_k)$; this enables us to produce trajectories with a starting time point of zero. Then, for each spatial segment seg_e we estimate the time t_e needed to reach that segment using the mean value of the records in that area $t_e = \frac{\sum time_{ij}}{|r_{ij}|}, \forall r_{ij} \in Route$ && $lat_{ij}, lon_{ij} \in seg_e$. This provides us a temporal estimation of a specific bus route to reach a spatial segment.

Definition 4 **Traffic Delay.** $delay_{ij}$ represents the difference of the estimated time to reach a segment (using Definition 3) compared to the actual time that the bus reaches. Hence, the $delay_{ij}$ for each new record r_{ij} is computed by the estimated time t_e to reach the spatial segment seg_e, where r_{ij} resides, compared to the time that has passed since the initiation of the trajectory, defined as:

$$delay_{ij} = t_e - (time_{ij} - \min(time_{mn} \forall r_{mn} \in traj_k))$$

Definition 5 **Minimum Bounding Rectangle (MBR).** Given a set of points P we define as a Minimum Bounding Rectangle (MBR) the minimul rectangular area that contains all points P.

Definition 6 **MBR Limits.** We define as MBR limits the spatial points, $p_1 : (lat_1, lon_1)$, $p_2 : (lat_2, lon_2)$ that reside at the bottom left and the top right upper corner respectively of the MBR and thus bound the area included.

Definition 7 **Spatiotemporal Delay Query.** We define as Spatiotemporal Query $Q(S_q, T(t_k, t_l))$ the query that computes the average delay within a spatial area S_q represented as an MBR, within a time window $T(t_k, t_l)$ we are interested in. The query has the following form:

$$Q(S_q, T(t_k, t_l)) = \frac{\sum delay_{ij}}{|r_{ij}|}, \forall r_{ij} \text{ s.t. } (lat_{ij}, lon_{ij}) \in S_q \;\&\&\; time_{ij} \in [t_k, t_l]$$

2.3. Overview

In this section we discuss the design of the sTREET tree structure. sTREET is optimized for real-time spatiotemporal queries that provide aggregated results over varying spatial regions within a time interval. The focus is on city streets where buses are used as speed probes, thus the range queries could also consider specific bus lines. For this reason, sTREET is primarily organized according to a spatial index, with temporal predicates, and then it is further divided based on the bus lines that reside in the spatiotemporal regions.

The basic storage layout of sTREET is presented in figure 1. As can be observed the root node preserves the spatiotemporal bounds along with the amount of total points preserved in the structure. At the second layer the nodes preserve their identifier, the spatial and temporal range for which they are responsible along with other information that includes the list of the node's children (set L), the amount of entries, and the parent node. We also provide an index for this layer that allow us to reduce the search

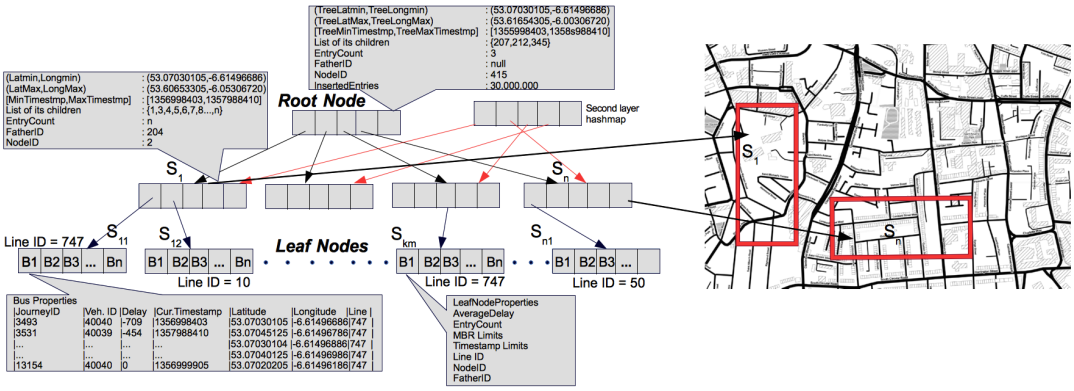

Figure 1. sTREET Tree.

space by several orders of magnitude as we show in our experiments, since we avoid sequential searches over the leaf nodes. The expansion procedure of our tree follows the principles of those in Trajectory Bundle Trees, in other words, expanding from left to right and up. This means that we can have several layers between the root node and the second layer(the layer above the leafs). Finally we divide these nodes based on the bus lines indexed at the nodes. Thus each leaf preserves the information retrieved from the buses in a specific line and also maintains the following information: the aggregated average delay computed based on all records in the leaf, the total entries in the leaf, the MBR limits of the points, the temporal limits of the included points and the identifiers of the line, the node and the parent of the node.

3. The sTREET Data Structure

The sTREET data structure is optimized to efficiently store and index spatio-temporal bus data so that by tracking bus locations and analyzing historical data, we can easily identify traffic delays and congestion. Using a set of optimizations, our data structure is capable of achieving minimal latency when looking up spatiotemporal data and answering real-time queries.

3.1. sTREET: TRavEl Estimation Trajectory Tree-Design

There has been a large body of work on indexing spatio-temporal data, where each of them focuses on a different aspect of the problem proposing a corresponding solution [9]. Perhaps the closest schemes to our approach are the R-trees and the STR-trees. The indices have similarities in organizing data into leaves, with similar insertion and splitting strategies, but with different focus on the data stored. In our approach, we make use of Trajectory Bundle trees [9] which are derived from R-trees [6]. They are designed to index trajectories over geographic regions, using rectangular boxes, known as minimum round boxes (MRB). They index the users' trajectories by storing the trajectory segments. In every leaf node, M segments (which is the fanout) are stored by preserving the constraint of trajectory discrimination. Our approach takes advantage of the benefits of TB-trees and we also optimize the tree in order to be able to retrieve the results under time constraints.

We developed a novel data structure the goal of which is to store trajectories of bus lines while offering a large range of spatiotemporal and navigational queries to perform. The data structure relaxes the constraint of the continuity of trajectories as presented in traditional TB-trees, but keeps the unique reference of a node to just one trajectory. Moreover, the data structure provides us with historical data about a specific line and its buses that can be derived from time window limits stored.

3.1.1. Tree Data Structure. All nodes of the data structure despite the layer they belong to, share some characteristics. Every node is stored in the hashmap with ⟨*Node ID, Node Object Reference*⟩ as key-value pair. We use hashmaps because it is known they have fast retrieval time and this is what we exploit in both our indexing scheme and the query processing. In the following subsections we present the design principles of our data structure.

Internal Nodes. The internal nodes of our data structure keep the same philosophy and design of those that were presented in R-trees [9]. They keep the fanout policy that triggers the splitting process, and for that reason they have a maximum capacity of leaf nodes they can hold underneath. Moreover, they store the minimum and the maximum values of GPS points probed by the buses and also the corresponding minimum and the maximum timestamps. These characteristics are used for denoting the coverage area in terms of both geospatial and time window parameters. The current number of inserted leafs and information of the node's level are also kept. Upon insertion of a new bus entry, the process of creating a new leaf, a "splitting" process which is similar to the one of the traditional TB-trees, will be triggered. It will update all the necessary information (*MRR limits and timestamp limits*) up to the root of the tree.

Secondary Layer. In our work, we give an advantage to the internal nodes belonging to the second layer (*i.e.*,

nodes on the next level above the leafs). We keep a reference of these nodes to a secondary index (a hashmap), in order to have fast access for our geospatial and time windows queries. With this option, we reduce the search range for any query that has to do with geospatial and time window constraints and in our experimental evaluation we illustrate the benefits of this approach.

Leaf Nodes. In our indexing approach, leaf nodes are used to store all the necessary information about the buses that travel along a specific line with a specific line Id. The number of buses (*i.e.*, the number of different combinations of $\langle lineId_{ij},\ vehicleJourneyId_{ij},\ vehicleId_{ij}\rangle$) has the same fanout as in internal nodes(for the children nodes respectively). Despite the fact that in traditional TB-trees, leaf nodes of the same trajectory are connected through a double linked list on the leaf level, and thus, one could iterate over the whole trajectory just using the pointers from one to another, in our approach we do not keep such list. Nodes belonging to the same line id can be easily found by just accessing the hashmap, as our goal is to achieve better results through the fetch procedure. Each leaf node has a field referring to the *node ID* which is unique. Moreover another field referring to the $lineId_{ij}$ is also used to apply the trajectory discrimination presented in TB-trees(in our case we have line discrimination). In the case that there is no available space to store a bus record, our method returns an error code that will trigger the process of adding a new leaf node, with a splitting strategy following if needed. Finally, each leaf node keeps the MBR and the timestamp limits of buses underneath.

3.1.2. Insertion and Querying in sTREET. Processing Queries. To process a spatiotemporal query $Q(S_q, T(t_k, t_l))$, sTREET first performs a lookup on the index to retrieve all nodes in the second layer of the tree that satisfy the spatiotemporal constraints of a given query $Q(S_q, T(t_k, t_l))$. The system then uses these nodes to fetch information from the leaf nodes belonging to nodes of the second layer. Taking into consideration that hashmaps provide fast retrieval times for any object, we benefit from this feature as we don't have to search linearly all leaf nodes, but use only nodes of the second level. This produces a set of records r_{ij} that satisfy the query and which are used to produce the final query results. To produce our results, we iterate only among the retrieved records and get the necessary relevant information for the given query, such as the Spatiotemporal Delay in the area bounded by a specific retrieved leaf node.

Handling Insertions. Each record r_{ij} is processed by sTREET that traverses through the nodes based on the spatiotemporal characteristics of r_{ij} $(lat_{ij}, lon_{ij}, time_{ij})$ and based on its $lineId_{ij}$ and can be either inserted to an existing leaf when it is not full or we might need to instantiate a new leaf to insert the record. Moreover, we update the information preserved in the parent nodes of the leaf to be consistent with the newly inserted node and we also update our index. The details of insertion into nodes and updating the indexes are described in detail in the following subsection.

Insertion - Splitting Algorithm. Our data structure inherits some principles from the insertion algorithm of the traditional TB-trees. In our case we do not want to descend the whole tree, until we find the last level of inner nodes. We refer to the nodes using the $lineId_{ij}$ as the key and we find the first available node. If such node does not exist, we create it and then we check if there is need to trigger the splitting process. If such a node exists, then we check whether the combination $\langle lineId_{ij},\ vehicleJourneyId_{ij},\ vehicleId_{ij}\rangle$ also exists. If so, we update the bus element timestamp limits, the mbr limits, the current delay and we also increase the number of points indexed by this node. Else, we create a new bus element and we update the node's limits appropriately. We still take into consideration the rightmost policy. In other words, in order to add a new leaf node, we are searching for the rightmost available position in the parent node. In case that such position is not available, we find the rightmost parent node (node that belongs to the second layer), we add the leaf node and update the spatiotemporal limits appropriately. Following this insertion rule, one can observe that our tree structure will expand from left to right and from bottom to upper, in all levels, and splitting processes will be triggered as needed.

3.2. Travel Time Estimation

In this section, we present how we exploit bus travel information stored in the sTREET data structure to infer travel times on a road network in real-time. Given a route $A \rightarrow B$ described with a set of non-overlapping MBRs S_q, that a user aims to travel within a time window $[t_k, t_l]$. Our goal is to derive the travel time needed for route $A \rightarrow B$, within the specified time interval using the information provided by the buses that cover these segments. The amount of time t_q needed to pass a regional space S_q can be computed using historical data. So the total time needed to travel between two points A, B is computed as the sum from all segments S_q of the estimated time needed to cross the segment t_q added with the average delay in location S_q within time $[t_k, t_l]$, extracted from the real-time bus information from query $Q(S_q, T(t_k, t_l))$, as: $\sum_{S_q \in A \rightarrow B}(t_q + Q(S_q, T(t_k, t_l)))$

4. Experiment Evaluation

In this section we present the experimental setup and performance results of our approach.

4.1. Experimental Setup

We have implemented the sTREET data structure using Java 1.8. Our results were produced on a quad-core Pentium 3.40Ghz with 16GB of RAM and a 1TB 7200RPM drive, running Windows 7. For every experiment we conducted, we configured a Java Virtual Machine with a maximum of 6 gigabytes of memory. The dataset upon which the experiments

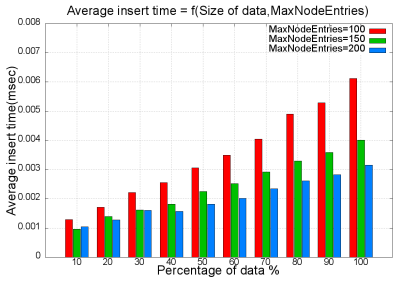

Figure 2. Average insert time as a function of the percentage of data size and MaxEntries.

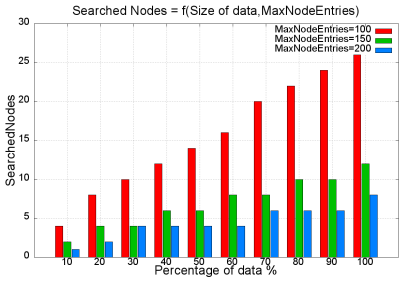

Figure 3. Searched nodes as a function of the percentage of data size for a large window query.

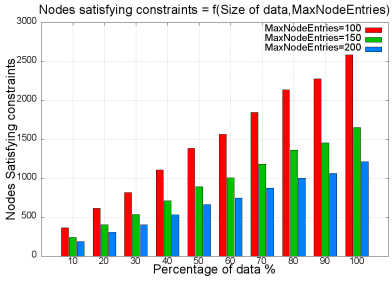

Figure 4. Nodes that satisfy the constraints.

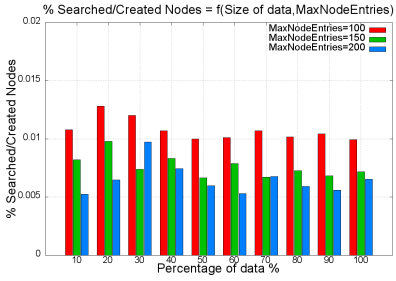

Figure 5. Percentage of searched nodes vs created nodes.

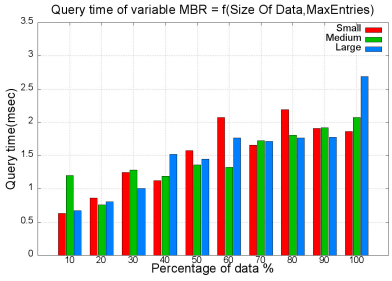

Figure 6. Query time as a function of the percentage of data size for Variable MBR sizes.

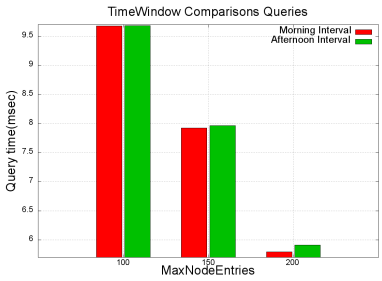

Figure 7. Query time for different MaxNodeEntries.

were conducted was a dataset from the INSIGHT project[1] providing Dublin Bus GPS data from Dublin City Council that contains more than 30 million GPS points [1]. In order to be able to illustrate the benefits of our indexing scheme, we have set and collected a set of performance metrics: *1) Average insert Time*, *2) Tree Expansion*, *3) Nodes Accessed* and *4) Query Time*. Our presented results are averages from 3 runs.

4.2. Experimental Results

4.2.1. Average Insertion Time experiment. The purpose of this experiment is to illustrate the scalability of our data structure. It was conducted by adding 27 days of bus data from our dataset, and we measured the insertion time needed in regard to the $MaxNodeEntries(fanout)$ parameter and the size of data. In figure 2 we can observe that the $Average\ insert\ time$ behavior when the percentage of data grows until day 27. For 50% of data inserted the time is slightly above 0.003 msec whereas for 100% of data inserted it may need 0.003-0.006 msec. We can also observe from the figure that the $fanout$ parameter significantly affects the average insertion time.

4.2.2. Tree Expansion and Node Accesses. Once the data were indexed in the sTREET, we performed another experiment based on the Spatiotemporal Query $Q(S_q, T(t_k, t_l))$ with S_q being the whole region indexed and t_k, t_l, were set as the minimum and maximum timestamp, so as to measure the amount of second level nodes searched for a given query and how many leaf nodes satisfied our constraints. Figure 3 shows that we needed to search up to 5 nodes for 10% of data where we needed to search between 8-26 nodes for 100% of the dataset. In figure 4 we can see how many leaf nodes satisfy our constraints. For a 10% of data under 500 nodes satisfy our constraints whereas for the full dataset 1300 to 2600 nodes satisfy our constraints. Thus, we state that the search space is minimal and reduced by orders of magnitude, compared to the naive approach of searching all leaf nodes to detect the ones that satisfy the criteria. Another interesting metric is the ratio of nodes being searched towards those created. Figure 5 illustrates the percentage of nodes to be searched towards those created. For 10% of data, this percentage varies from 0.005% to 0.011%, while for the full dataset it varies from 0.007% to 0.010%. The figure shows that this ratio remains constant despite the number of nodes created.

4.2.3. Query Time experiment. For this set of experiments, we performed queries of the form $Q_k(S_q, T_{i,j})$ where we variate the S_q and the $T_{t_k,tl}$.

Variable MBR experiment. We also measured the performance of our data structure by adjusting the MBR limits while keeping the same time window limits. For this experiment, we computed the time needed to perform the query taking as a fixed parameter the parameter $MaxNodeEntries = 150$. The MBR Limits used were a) a small one, an MBR Limit that only one second level node could store, b) a medium one, an MBR Limit that takes into account more than one second level layer nodes, and finally c) a large one, an MBR Limit that would fetch the results from the whole region underneath. Figure 6 shows

1. INSIGHT - http://www.insight-ict.eu/

that for 10% of data we may need from 0.7-1.2msec to fetch the final result whereas for the full dataset we need 1.8-2.7msec. Thus, despite the variation of the MBR limits, the amount of time needed to answer a query is very small.

Time window experiment. We also conducted an experiment to illustrate the behavior of our data structure when the parameter that changes is the time window. We performed our $Spatiotemporal\ Average\ Delay$ query with variable time window limits. We set the S_q as the entire area indexed, and we use a a time window pair list $Ls\langle T_{i,j}\rangle$, that contains the time window pairs to query our data structure. The list $Ls\langle T_{i,j}\rangle$ we used in our experiment contained all week days and a time interval of (i) 8-11 am for the Morning and (ii) a time interval of 2-6 pm for the Afternoon. In figure 7 we compare the amount of time needed for both intervals in connection with $MaxNodeEntries$. As can be observed, the query time for averaging delays from a whole week takes 3-11msec and it strongly depends on the $MaxNodeEntries$ parameter we choose.

5. Related Work

The problem of indexing spatiotemporal data is not new and several algorithms have been proposed in the literature. In our previous work [4] we have developed an enhanced version of trajectory bundle trees. It was designed to store trajectory segments derived from taxis, in a similar approach with trajectory bundle trees, but by using different types of keys apart from $trajectoryId$, such as the $userId$ of each driver. However, in this work, we have extended our approach in order to optimize the retrieval process and reduce the information kept in each node. Designed to support real-time queries, we have enhanced our tree structure by adding a secondary index that minimizes the search space and the time needed to access the nodes.

In [9], the Trajectory Bundle trees data structure was presented. The basic idea is that they keep segments of trajectories with spatiotemporal similarities between trajectories and preserve trajectory discrimination. However, they cannot preserve information about the current delays on the road network as in our approach, and furthermore they would need a larger number of processing steps to get the $Spatiotemporal\ Average\ Delay$. Other approaches such as TrajStore [5] use a quad-tree in order to index points and trajectories. They segment and index trajectory segment that lie spatiotemporaly near blocks of disk, whereas in our approach we organize our index in memory. However, we are able to index trajectories and their points as data comes in streams. In [10] the authors propose an algebraical approach using space decomposition to estimate travel time, whereas our work focuses on a minimal latency data structure to accomplish the same goal. In [11] they propose a graph approach in order to estimate the optimal cost route but they do not use any kind of index in order to support other types of queries.

6. Conclusions

In this paper we have presented sTREET, our novel tree-based structure that is designed to index large and dynamic spatiotemporal trajectories. sTREET is optimized to use for travel time estimation using real-time information produced from buses equipped with automatic vehicle location systems. Our detailed experimental results show that sTREET is practical, can effectively retrieve spatiotemporal data produced in real-time, meet application demands, and is highly efficient due to its minimal response times.

Acknowledgments

This research has been co-financed by the European Union (European Social Fund ESF) and Greek national funds through the Operational Program Education and Lifelong Learning of the National Strategic Reference Framework (NSRF) - Research Funding Program: Thalis-DISFER, Aristeia-MMD, Aristeia-INCEPTION Investing in knowledge society through the European Social Fund, the FP7 INSIGHT project and the ERC IDEAS NGHCS project.

References

[1] Dublin bus data. http://dublinked.com/datastore/datasets/dataset-304.php.

[2] P. K. Agarwal and C. M. Procopiuc. Advances in indexing for mobile objects. *IEEE Data Eng. Bull.*, 25(2):25–34, 2002.

[3] O. Andersen, C. S. Jensen, K. Torp, and B. Yang. Ecotour: reducing the environmental footprint of vehicles using eco-routes. In *MDM*, volume 1, pages 338–340, Milan, Italy, June 2013.

[4] I. Boutsis, D. Tomaras, and V. Kalogeraki. Towards real-time emergency response using crowdsourcing. In *PETRA*, Rhodes, Greece, May 2014.

[5] P. Cudre-Mauroux, E. Wu, and S. Madden. Trajstore: An adaptive storage system for very large trajectory data sets. In *ICDE*, pages 109–120, Long Beach, CA, USA, March 2010.

[6] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *SIGMOD Conference*, pages 47–57, 1984.

[7] M. Haridasan, I. Mohomed, D. Terry, C. A. Thekkath, and L. Zhang. Startrack next generation: A scalable infrastructure for track-based applications. In *OSDI*, pages 409–422, Vancouver, Canada, October 2010.

[8] D. Kinane, F. Schnitzler, S. Mannor, T. Liebig, K. Morik, J. Marecek, B. Gorman, N. Zygouras, Y. Katakis, V. Kalogeraki, and D. Gunopulos. Intelligent synthesis and real-time response using massive streaming of heterogeneous data (insight) and its anticipated effect on intelligent transport systems (its) in dublin city, ireland. In *ITS*, Dresden, Germany, November 2014.

[9] D. Pfoser, C. S. Jensen, Y. Theodoridis, et al. Novel approaches to the indexing of moving object trajectories. In *VLDB*, pages 395–406, Cairo, Egypt, September 2000.

[10] Y. Wang, Y. Zheng, and Y. Xue. Travel time estimation of a path using sparse trajectories. In *SIGKDD*, New York, NY, August 2014.

[11] B. Yang, C. Guo, C. S. Jensen, M. Kaul, and S. Shang. Multi-cost optimal route planning under time-varying uncertainty. In *ICDE*, Chicago, IL, USA, April 2014.