Torch: A Search Engine for Trajectory Data

Sheng Wang
RMIT University
sheng.wang@rmit.edu.au

Zhifeng Bao
RMIT University
zhifeng.bao@rmit.edu.au

J. Shane Culpepper
RMIT University
shane.culpepper@rmit.edu.au

Zizhe Xie
National Key Laboratory for Novel Software Technology, Nanjing University
forrest0402@smail.nju.edu.cn

Qizhi Liu
National Key Laboratory for Novel Software Technology, Nanjing University
lqz@nju.edu.cn

Xiaolin Qin
Nanjing University of Aeronautics and Astronautics
qinxcs@nuaa.edu.cn

ABSTRACT

This paper presents a new trajectory search engine called Torch for querying road network trajectory data. Torch is able to efficiently process two types of typical queries (similarity search and Boolean search), and support a wide variety of trajectory similarity functions. Additionally, we propose a new similarity function LORS in Torch to measure the similarity in a more effective and efficient manner. Indexing and search in Torch works as follows. First, each raw vehicle trajectory is transformed to a set of road segments (edges) and a set of crossings (vertices) on the road network. Then a lightweight edge and vertex index called LEVI is built. Given a query, a filtering framework over LEVI is used to dynamically prune the trajectory search space based on the similarity measure imposed. Finally, the result set (ranked or Boolean) is returned. Extensive experiments on real trajectory datasets verify the effectiveness and efficiency of Torch.

ACM Reference Format:
Sheng Wang, Zhifeng Bao, J. Shane Culpepper, Zizhe Xie, Qizhi Liu, and Xiaolin Qin. 2018. Torch: A Search Engine for Trajectory Data. In *SIGIR '18: The 41st International ACM SIGIR Conference on Research and Development in Information Retrieval, July 8–12, 2018, Ann Arbor, MI, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3209978.3209989

1 INTRODUCTION

Equipped with Global Positioning System (GPS) devices, vehicles can now use various location-based services such as Waze to navigate complex road networks. By sampling a series of points of a vehicle path along the road network at fixed time intervals, a trajectory can be recorded using three features: 1) Network-constrained, as most trajectories travel in a fixed road network [19]; 2) Varying sample rates – every trace from a same path can be sampled to a different number of points; 3) Errors as a result of low GPS accuracy. A wide variety of *trajectory search* queries [3, 10, 17–19, 32, 33, 36, 38] have been proposed over the years to support various location-based services such as transportation monitoring and planning [47], or ranked retrieval, as shown in Example 1.

SIGIR '18, July 8–12, 2018, Ann Arbor, MI, USA

ACM ISBN 978-1-4503-5657-2/18/07…$15.00
https://doi.org/10.1145/3209978.3209989

Example 1. *To monitor vehicles passing through Manhattan, a user would issue a* ***range query****. To monitor all cars that use Wall Street, a* ***path query*** *[19] would be issued. Further, a* ***strict path query*** *would identify every vehicle that traverse all of Wall Street. These three kinds of queries are* ***Boolean trajectory search queries****. Given a trajectory, a* ***top-k trajectory similarity search query*** *returns the k highest ranked trajectories based on a similarity metric [12, 47], which can be used to investigate driving habits [5], cluster trajectories to discover popular routes [46], or search over all of the trajectories which have a pre-specified trajectory as a k nearest neighbor – a task used in transportation route planning [42].*

It is desirable to support all of these search queries in a single *trajectory search engine*. Furthermore, there are several commonly used similarity measures to determine the relevance between two trajectories (details in Section 2), each with its own merits. However, the effectiveness of these similarity measures has only been evaluated in domain specific search scenarios such as time series [4, 45]. The lack of an efficient trajectory search engine capable of supporting different similarity measures makes it very difficult to carry out careful effectiveness evaluations. For example, existing trajectory systems [2, 7, 41] only support trajectory storage and simple querying, but cannot perform top-k similarity search. In this paper, we devise a trajectory search engine capable of supporting a richer set of complex queries and similarity measures.

The two primary goals of a search engine are effectiveness (quality) and efficiency (speed) [6]. These two desiderata are often in tension with each other. Due to the varying sampling rates of trajectories, existing similarity measures are rarely efficient and effective enough to search even modest sized trajectory collections (Details in 4.1). Moreover, the effectiveness evaluation of top-k similarity search over trajectories currently rely on improving the classification precision of time series data. Determining whether a similarity measure can classify a set of trajectories into the corresponding labeled ground-truth, such as by Chen et al. [4], is not a sufficient approach when evaluating the quality of a search engine.

To achieve scalable efficiency, a space-efficient index representation and processing framework is crucial, but existing indexes for trajectory search [28, 31, 38] rely on an R-tree [14], which stores all points from the raw trajectories. They often require a prodigious amount of space in order to accelerate search, and the pruning method originally devised for point search is not effective for trajectory search as many of the similarity measures are non-metric[1].

[1] https://en.wikipedia.org/wiki/Metric_(mathematics)

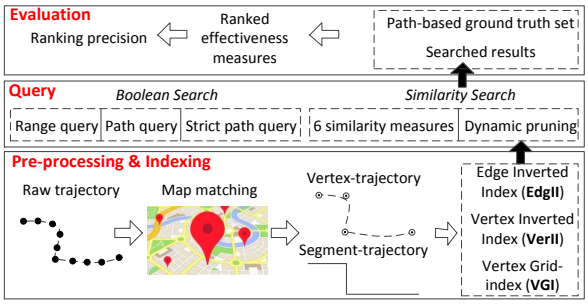


Figure 1: An overview of Torch.

Inspired by these observations, we design and implement Torch, that contains the three modules shown in Figure 1. First, pre-processing derived from *map matching* [22] projects raw trajectories to a succinct path, where each mapped trajectory is represented by a list of vertices and edges in a road network. Based on this new representation, a new similarity measure (LORS) is developed which can significantly improve the effectiveness and efficiency of similarity search in real data collections. The processing speed is further improved by using a lightweight edge and vertex index (LEVI), which is highly compressable using standard integer list compression techniques. Top-k search using both LORS and existing similarity measures can be performed efficiently using the unified dynamic pruning algorithm presented in this paper. In addition, Boolean trajectory search can be performed efficiently in a manner similar to Boolean processing in Information Retrieval systems [9]. Finally, the search results from all similarity measures can be evaluated using a new path-based ground truth set.

In summary, we make the following contributions:

- We present Torch, a trajectory search engine which integrates pre-processing, indexing, querying and evaluation.
- We propose a novel trajectory similarity measure – *longest overlapped road segments* (LORS) based on the overlapped segments between query and trajectory data (Section 4).
- We propose a unified index – LEVI with compression (Section 5), and an efficient search paradigm (Section 6) to support similarity and Boolean search over trajectories.
- We employ Torch to conduct a comprehensive efficiency and effectiveness evaluation of similarity measures using two real taxi trajectory datasets (Section 7).

2 RELATED WORK

Boolean Trajectory Search. A Boolean trajectory search includes three main types of queries. The first one is a *Range Query* (RQ) [19, 21, 32, 33, 35] that finds all trajectories located in a spatial region. The second is a *Path Query* (PQ) which retrieves the trajectories that contain any edge of the given path query. The third is a *Strict Path Query* (SPQ) [17–19, 33] which finds all trajectories that strictly follow the entire path from beginning to end. Interestingly, the path query and strict path query share many commonalities with disjunctive (OR) and conjunctive (AND) Boolean queries [23]. The main difference is that an SPQ is order-sensitive.

Trajectory Similarity Measures. As a trajectory can be viewed as a time series, many similarity measures originally designed for time series search can be deployed in trajectory similarity search [52], such as Dynamic Time Warping (DTW) [50], Longest Common Subsequence (LCSS) [25, 39], and Edit Distance on Real sequence (EDR) [3]. These similarity measures were designed to make search more robust against local time shifting and noise. An effectiveness evaluation of these similarity measures over time series data [4] was performed by Wang et al. [45].

As a trajectory can also be viewed as a *geometric curve*, similarity measures between curves can also be used for trajectory similarity search [47]. For example, Hausdorff distance [27, 32] measures how far two subsets of a metric space are from each other. Fréchet distance [1] extends the Hausdorff distance to account for location and ordering of the points along the curves.

By revisiting the above similarity measures in two different contexts, we make three observations. (1) The similarity computation is a procedure to match points without violating an order constraint, and we refer to these measures as a *point-based similarity measure*. (2) To the best of our knowledge, the effectiveness of common similarity measures for both time series and geometric curves have not been evaluated for real vehicle trajectory data. (3) These similarity measures are sensitive to sampling rate [40], as sampling more or fewer points will affect the overall similarity.

Pruning for Similarity Search. The R-tree [14] is a popular spatial index for point data which can be extended to support pointwise trajectory data. Trajectory datasets like T-drive [51] can have millions of points, and an R-tree has to manage a huge number of maximum bounding rectangles (MBR), which is not space efficient in practice. A simple grid index can be a more useful index structure in such scenarios [5, 43]. For search, most of the existing approaches use MBR-based pruning, as MBR intersection with the query trajectory can be used to prune out irrelevant trajectories [16]. Bounded computations are another method used to accelerate search [30, 39]. This is achieved by estimating the upper bound of the similarity score and comparing it with the current top-k result set to determine if the exact distance must be computed. This approach is often referred to as *early abandoning* [30].

3 PROBLEM DEFINITION

3.1 Data Model

Definition 1. ***(Point)*** *A point $p = \{p.lat, p.lng, p.t\}$ contains the spatio-temporal information, which includes latitude $p.lat$, longitude $p.lng$ and time-stamp $p.t$.*

Definition 2. ***(Raw Trajectory)*** *A trajectory T of length n is in the form of $\{p_1, p_2, \ldots, p_n\}$, where each p_i is a point.*

Definition 3. ***(Road Network)*** *A road network is a graph $G = (V, E)$, where V is a set of vertices v representing the intersections and terminal points of the road segments, and E is a set of edges e representing road segments.*

Definition 4. ***(Path)*** *A path P is composed by a set of connected road segments $e_1 \rightarrow e_2 \rightarrow \ldots \rightarrow e_{n+1}$ in G.*

The object that passes the path P in G can be sampled as different raw trajectories $S_P = \{T_1, T_2, \cdots, T_m\}$. S_P may be lossy, as it may not store all of the information from the original path P. So, it is essential to transform S_P to the original path P, especially for trajectory-based applications [22]. This is commonly referred to as *map matching* [22, 26], and defined as:

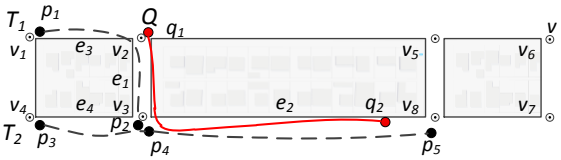


Figure 2: Original routes and sampled raw trajectories over a road network with edge e and vertex v.

Definition 5. ***(Map-Matched Trajectory)*** *Given a raw trajectory T and a road network G, the map-matched segment-trajectory $\overline{T}$ is a set of connected road segments projected from GPS points in T onto the road segments in G, such that $\overline{T} : e_1 \rightarrow e_2 \rightarrow \ldots \rightarrow e_n$. Further, the map-matched vertex-trajectory $\ddot{T}$ is a set of connected graph vertices in G, such that $\ddot{T} : v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_{n+1}$, and $e_i = (v_i, v_{i+1})$. This mapping procedure is denoted as $T \xrightarrow{G} (\overline{T}, \ddot{T})$.*

Figure 2 shows two raw trajectories T_1 and T_2, and a query Q, where a solid dot represents a point of the trajectory. $T_1 = \{p_1, p_2\}$ and $T_2 = \{p_3, p_4, p_5\}$ can be mapped to the path (dotted line) in a road network (hollow dots) and represented by edges $\{e_3, e_1\}$ and $\{e_4, e_2\}$, or by vertices $\{v_1, v_2, v_3\}$ and $\{v_4, v_3, v_8\}$, respectively. Several open-source libraries or APIs are also available.[2,3]

3.2 Query Model

Definition 6. ***(Boolean Trajectory Search (BTS))*** *Given a trajectory database $\mathcal{D} = \{T_1, \ldots, T_{|D|}\}$ and query constraint Q, a Boolean Trajectory Search retrieves the trajectories in three ways:*

$$RQ(Q_r) = \{T \in \mathcal{D} | \exists p_i \in T (p_i \in Q_r)\} \quad (1)$$

$$PQ(Q_p) = \{T \in \mathcal{D} | \exists e_i \in \overline{T}, e_j \in Q_p (e_i = e_j)\} \quad (2)$$

$$SPQ(Q_p) = \{T \in \mathcal{D} | \exists i, j (\overline{T}_{ij} = Q_p)\} \quad (3)$$

where $\overline{T}_{ij} = \{e_i, e_{i+1}, \cdots, e_j\}$ is the sub-trajectory of $\overline{T}$, Q_r is a rectangular region, and Q_p is a path in G.

Definition 7. ***(Trajectory Similarity Search (TSS))*** *Given a trajectory database $\mathcal{D} = \{T_1, \ldots, T_{|D|}\}$ and query trajectory $Q = \{q_1, q_2, \cdots, q_{|Q|}\}$, a Top-k Trajectory Similarity Search retrieves a set $\mathcal{D}_s \subseteq \mathcal{D}$ with k trajectories such that: $\forall T \in \mathcal{D}_s, \forall T' \in \mathcal{D} - \mathcal{D}_s, \hat{S}(Q, T) > \hat{S}(Q, T')$.*

Here, $\hat{S}$ is the similarity function between two trajectories. Given two raw trajectories T_1 and T_2 or the map-matched vertex-trajectories $\ddot{T}_1$ and $\ddot{T}_2$, the similarity between $\hat{S}(T_1, T_2)$ and $\hat{\ddot{S}}(\ddot{T}_1, \ddot{T}_2)$ can be computed using most existing functions, such as DTW [50], LCSS [39], EDR [3], Hausdorff [27] and Fréchet distance [1].[4]

4 A SEGMENT-BASED SIMILARITY MEASURE

4.1 Point-based Similarity

LCSS [39] measures the common subsequence of two trajectories, similar to the *longest common sub-string matching* problem, where two points are treated as matched when they have a distance less than a pre-defined threshold τ. For example in Figure 2, given a query Q, the LCSS similarity will be $\hat{S}(Q, T_1) = m(q_1, p_1) + m(q_2, p_1)$, $\hat{S}(Q, T_2) = m(q_1, p_4) + m(q_2, p_5)$, where m denotes the matching relationship between points that have a distance less than τ. We can find that the point pairs do not match well for raw trajectories when two points far apart are chosen to match, such as p_1 and q_1. After the map matching transformation, the points are calibrated by edges and vertices, and point-matching can be used to identify similar trajectories. For example, the similarities of trajectories: $\ddot{Q} = \{v_2, v_3, v_8\}$, $\ddot{T}_1 = \{v_1, v_2, v_3\}$, $\ddot{T}_2 = \{v_4, v_3, v_8\}$, are computed as $\hat{\ddot{S}}(\ddot{Q}, \ddot{T}_1) = m(v_2, v_2) + m(v_3, v_3) = 2$, $\hat{\ddot{S}}(\ddot{Q}, \ddot{T}_2) = m(v_3, v_3) + m(v_8, v_8) = 2$, where each matching m has a reward of 1.

However, it can be easily observed that T_1 and T_2 are not different from Q in term of similarity, while T_2 shares a longer overlap with Q than T_1 and should be more similar to Q. Such an overlapping relationship of road segments is not captured in any of the existing point-based similarity measures. Moreover, the cost of computing point-based similarity measures is high, as the Euclidean distance computation needs to be performed for every matching point pair by accessing the locations of the points. In order to solve the above two problems, we propose a parameter-free similarity model based on segments rather than points.

4.2 Longest Overlapping Road Segments

Inspired by LCSS [39], we define the Longest Overlapping Road Segment (LORS) to measure the similarity between two map-matched segment-trajectories. Let $\overline{T}_1$ and $\overline{T}_2$ be two map-matched segment-trajectories of T_1 and T_2, where $\overline{T}_1 = (e_{11}, ..., e_{1n})$ and $\overline{T}_2 = (e_{21}, ..., e_{2m})$.

Definition 8. *(LORS) The Longest Overlapping Road Segment Similarity $\hat{\overline{S}}(\overline{T}_1, \overline{T}_2)$ is defined as:*

$$\hat{\overline{S}}(\overline{T}_1, \overline{T}_2) = \begin{cases} 0, & \text{if } \overline{T}_1 \text{ or } \overline{T}_2 \text{ is empty} \\ |e_{1n}| + \hat{\overline{S}}(\mathrm{H}(\overline{T}_1), \mathrm{H}(\overline{T}_2)), & \text{if } e_{1n} = e_{2m} \\ \max(\hat{\overline{S}}(\mathrm{H}(\overline{T}_1), \overline{T}_2), \hat{\overline{S}}(\overline{T}_1, \mathrm{H}(\overline{T}_2))), & \text{otherwise} \end{cases} \quad (4)$$

where $|e_{1n}|$ is the travel length of graph edge e_{1n}. $H(\overline{T}_1) = (e_{11}, ..., e_{1n-1})$ is the sub-trajectory of $\overline{T}_1$ minus the last point. With LORS, the length of a segment will be added to the overall similarity instead of a unit cost 1 in LCSS, which is more discriminative when distinguishing between similar trajectories. Moreover, LORS does not require a threshold τ, and avoids complex spatial computations as it does not need to compute the distance between two edges or vertices which must access additional positional information. Instead, LORS only needs to determine whether the edge IDs (edgID) of e_{1n} and e_{2m} are the same. The effectiveness and efficiency of LORS are explored further in Section 7.4.

Example 2. *As shown in Figure 2, a query Q has three points and two edges e_1 and e_2, the length of the two edges are 10 and 50, respectively. There are two trajectories T_1 and T_2 that intersect with e_1 and e_2. T_2 has a longer common segment with Q, and should be more similar to Q, and LORS can preserve this relationship, where $\hat{\overline{S}}(\overline{Q}, \overline{T}_1) = |e_1| = 10$, and $\hat{\overline{S}}(\overline{Q}, \overline{T}_2) = |e_2| = 50$.*

Property 1. *LORS is a similarity measure that is non-metric and robust to noise, and also can handle local time shifting in a collection.*

This property can be validated by observing a simple counter-example in Figure 2, $\hat{\overline{S}}(\overline{Q}, \overline{T}_1) = 10$, if $\hat{\overline{S}}(\overline{Q}, \overline{T}_2) = 50$ and $\hat{\overline{S}}(\overline{T}_1, \overline{T}_2) =$

[2] https://github.com/graphhopper/map-matching
[3] https://www.mapbox.com/
[4] Distance functions such as DTW and EDR can be easily converted to a similarity function using renormalization.

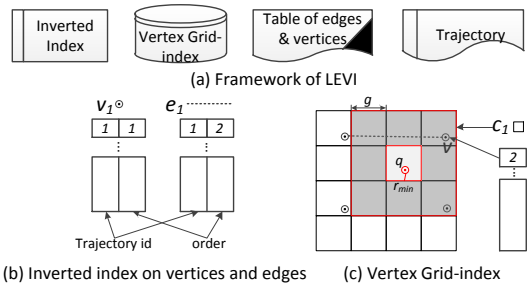


Figure 3: Index on the vertices and edges.

0, then $|\hat{\overline{S}}(\overline{T}_1, \overline{T}_2) - \hat{\overline{S}}(\overline{Q}, \overline{T}_1)| = |\hat{\overline{S}}(\overline{T}_1, \overline{T}_2) + \hat{\overline{S}}(\overline{Q}, \overline{T}_1)| < \hat{\overline{S}}(\overline{Q}, \overline{T}_2)$. Hence LORS does not obey the triangular inequality, and is non-metric. The local time shifting property can also be observed from the matching relationships. LORS is robust to noise as "noisy" edges cannot contribute to the final similarity score.

It is worth noting that a map-matched trajectory must have at least one common edge with the query in order for LORS to be non-zero, so a baseline which finds all possible candidates is to find all trajectories which share at least one edge with the query. Hence, an inverted index can be employed to support this search. Storing only the identifiers for trajectories is not sufficient when supporting the LORS similarity measure as local time shifting is possible, and the order of each point in the trajectory must therefore be considered.

5 LIGHTWEIGHT EDGE AND VERTEX INDEX

To support efficient search on LORS and other commonly used similarity measures, we propose a lightweight edge and vertex index (LEVI). Figure 3(a) describes the two indexes (inverted index and Grid-index), as well as the storage of vertices, edges and trajectories.

5.1 Inverted Lists on Edges and Vertices

Definition 9. ***(Edge Inverted Index (EdgII))*** *The inverted index I_e of an edge e stores the tuples $\{traID, order\}$ of all map-matched segment-trajectories $\overline{T}$ that overlap with e, where traID is the unique identifier of $\overline{T}$ and order is the position of the edge e in $\overline{T}$.*

In addition to storing traIDs in the inverted index, the order of edges in a trajectory is also required when computing the similarity with local time shifting. This is analogous to a Positional Inverted Index [49] in text retrieval. The main difference is that we do not need to store the frequency information which is essential for text retrieval similarity function such as TF·IDF [23]. Moreover, most trajectories are usually on a trip basis, and will not cross a road segment several times, and the frequency is rarely greater than 1. So, we use two lists to store the traID and order respectively. For example in Figure 4, $\overline{T}_{15}$, $\overline{T}_{31}$, $\overline{T}_{39}$ and $\overline{T}_{43}$ cross edge e_{32} in the position of 12, 1, 33, 28, respectively, then the positional inverted list is stored as $\{\{15, 12\}, \{31, 1\}, \{39, 33\}, \{43, 28\}\}$ ordered by $traID$.

For our new similarity function LORS, we will access the inverted index by edge for the road network, and only the trajectories that intersect the query edges can be candidates. To further support other point-based similarity measures, LEVI includes another index structure, the *vertex inverted index* (VerII) for each vertex similar to EdgII, which is denoted as I_v, Figure 3(b) shows the structure of VerII and EdgII, $\{1, 1\}$ means that a map-matched vertex-trajectory $\ddot{T}_1$ crosses vertex v_1 in the first position. Additionally, we maintain

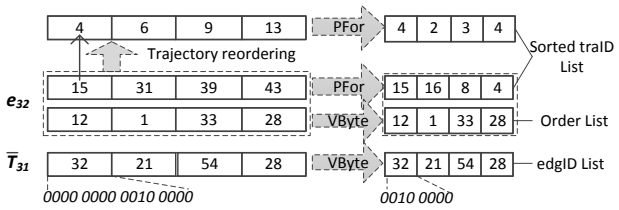


Figure 4: Compression and storage of a trajectory in EdgII.

a table (right of Figure 3(c)) to store the latitude and longitude of each vertex v, and the travel length of each edge e for the similarity computation, respectively.

5.2 Trajectory and Index Compression

Trajectory Compression. In most existing point-based measures such as DTW and EDR, each point in the candidate trajectory must be matched with a query point in the similarity computation, so it is critical to store the entire trajectory as compactly as possible. A map-matched trajectory $\ddot{T}$ and $\overline{T}$ composed of vertices and edges can be represented by a list of integers which are the IDs of the vertices and edges, and compressed. In Figure 4, $\overline{T}_{31}$ crosses edges $e_{32}, e_{21}, e_{54}, e_{28}$, so $\overline{T}_{31}$ is stored as $\{32, 21, 54, 28\}$ without sorting in order to maintain the original ordering. Similar to compressing the unsorted integer list, we use VByte [8, 24, 34] to compress the trajectories composed by unsorted identifiers of edges or vertices.

Inverted List Compression. As the inverted index is composed of posting lists of trajectory identifiers (traID), denoted by integers, the sorted list of integers can be compressed using efficient and effective compression techniques such as *delta encoding* [20]. Moreover, trajectory ordering information must be maintained and aligned with its trajectory identifiers. Recall that in the example in Figure 4 the inverted index can be divided into two lists: $\{12, 1, 33, 28\}$ which is unsorted; and $\{15, 31, 39, 43\}$ which is sorted to facilitate compression with delta encoding $\{15, 31-15, 39-31, 43-39\} = \{15, 16, 8, 4\}$ and VByte $12 = 0000000000001100 \rightarrow 1100$.

Trajectory Reordering. To further compress the inverted lists by reducing the d-gaps, reassigning *traID* can also have a significant effect. For example in Figure 4, the *traID* list $\{15, 31, 39, 43\}$ is reordered to $\{4, 6, 9, 13\}$, then the compression based on delta encoding $\{4, 2, 3, 4\}$ can save more space than the original $\{15, 16, 8, 4\}$. *traID* should be assigned such that trajectories sharing many vertices and edges are close to each other. A weighted graph over the original *traID* can be constructed such that there is an edge between two trajectories if they share a *verID* or *edgID*. The weight used is the LORS similarity between the two trajectories. Then, we use a state-of-the-art graph reordering technique [11] to reorder the trajectories. For a newly inserted trajectory, we can assign the traID incrementally and perform the reordering batchwise.

5.3 Vertex Grid-index for Range Querying

For real-distance based functions such as DTW, Hausdorff and Fréchet distance, any trajectory can be a candidate even when it does not intersect with the query at all. So, a spatial index such as an R-tree [14] or Grid-index [43] is required to index all the vertices in G. An *incremental range queries* is ran on the vertices of the query to identify all neighboring vertices, and then the inverted

index of vertices VerII is accessed to generate the IDs of candidate trajectories. Similar to existing work [43, 44] on searching trajectories using pointwise techniques, we use a Grid-index to index the vertices in the road network, and refer to it as a *Vertex Grid-index* (VGI), which is another important component of LEVI. Note that VGI is also compressible but is a small index in practice as even large cities such as Beijing have only 54, 406 total vertices. As shown in Figure 3(c), we build a sorted list for each cell c to store the verIDs of all the vertices located in c, and assign a unique code for each cell based on the Z-curve [44], then take delta encoding and compress.

VGI supports two search operators: a range query $RS(q, r_i)$ and an incremental range search $IRS(q, i, g)$ where g is the cell size, and i is the current search round. Here, $IRS(q, 0, g)$ returns the cell where the query is located. Figure 3(c) shows a VGI with a cell size of g, with two rectangles in the index. $IRS(q, 1, g)$ finds all of the vertices located outside of the small rectangle where q is located ($IRS(q, 0, g)$), also inside the larger rectangle – the annular region (grey area). Such an operator avoids repetitive scanning of vertices and accesses the search space by increasing a cell size g, Algorithm 1 is then used to compute similarity and do any necessary pruning.

6 PRUNING FOR SIMILARITY SEARCH

Boolean Trajectory Search (BTS) can be easily processed using LEVI. We first perform a range query in VGI, and access the inverted index of the edges that intersect with the range, or the vertices that are located inside the range. For trajectory similarity search (TSS), a naive algorithm would process every trajectory in the dataset and return the k best results, requiring an enormous number of I/O and similarity computations. Hence, effective pruning is essential for efficient similarity search.

6.1 Algorithm Overview

Similar to dynamic pruning strategies such as MaxScore [37] over posting lists from the Information Retrieval area, and the *Threshold algorithm* [13] from database area, our algorithm is composed of filtering (Line 2 to 7) and refinement (Line 18 to 23) based on LEVI, as shown in Algorithm 1.

To filter out non-qualified trajectories, the inverted indexes EdgII: I_e and VerII: I_v play an important role in every similarity measure (Line 4 and 7). The main difference is that accessing the inverted index is conducted once with LORS and multiple times by existing similarity measures which also use VGI. Termination occurs when the k·th result R_k in the result set R has a similarity greater than the upper bound for the remaining unprocessed trajectories UB (Line 10). The refinement step is based on bound reordering techniques, such as sorting of all of the candidate trajectories $T \in can$ by upper bound similarity $\hat{S}^{\uparrow}(Q, T)$,[5] then computing the true similarity through pivoting. Processing stops when the next upper bound is smaller than the k·th result (Line 22).

6.2 Pruning for LORS

Recall from Property 1 that LORS is non-metric, for which a tree-structure index such as an R-tree cannot be used for pruning. To this end, the main idea of our pruning is to filter the candidates

[5] In this section the trajectory T and query Q are all map-matched trajectories by default. We drop $\overline{T}$ and $\ddot{T}$ to distinguish from raw trajectories for the sake of convenience. Similarly, $\hat{S}$ can be $\hat{\overline{S}}$ or $\hat{\ddot{S}}$ as our algorithm supports both.

Algorithm 1: Trajectory similarity search

Input: $\{Q, k\}$: query, D: dataset, I: index, $\hat{S}$: similarity measure
Output: Top-k result set R

1 $can \leftarrow \emptyset$, $R \leftarrow \emptyset$, $i_{IRS} = 0$, $UB \leftarrow 0$;
2 **for** *every* $q \in Q$ **do**
3 **if** $\hat{S} =$ *LORS* **then**
4 $can \leftarrow can \cup I_e(q)$;
5 **if** $\hat{S} =$ *LCSS or* $\hat{S} =$ *EDR* **then**
6 $s \leftarrow RS(q, \tau)$;
7 $can \leftarrow can \cup \bigcup_{q' \in s} I_v(q')$;
8 Sort all the trajectories $T \in can$ by $\hat{S}^{\uparrow}(Q, T)$;
9 **if** $\hat{S}$ *is a real distance function* **then**
10 **while** $\hat{S}(Q, R_k) \geq$ UB **do**
11 **for** *every* $q \in Q$ **do**
12 $s \leftarrow IRS(q, i_{IRS}, g)$;
13 $can \leftarrow can \cup \bigcup_{q' \in s} I_v(q')$;
14 $i_{IRS} \leftarrow i_{IRS} + 1$;
15 Sort the trajectories $T \in can$ by $\hat{S}^{\uparrow}(Q, T)$;
16 $UB \leftarrow \hat{S}^{\uparrow}(Q, i_{IRS})$;
17 Choose the top-k and update R;
18 **while** $T_i \in$ can **do**
19 **if** $\hat{S} \neq$ *LORS* **then**
20 $T_i \leftarrow AccessFullTrajectory(D, T_i.id)$;
21 $Update(R, \hat{S}(Q, T_i))$;
22 **if** $\hat{S}(Q, R_k) \geq \hat{S}^{\uparrow}(Q, T_{i+1})$ **then**
23 break;
24 return R;

without computing the final similarity as the complexity of LORS is $O(mn)$, where m and n are the number of segments in Q and T, respectively. To avoid computing the real similarity one by one, we compute an upper bound which has a linear complexity cost, and check whether it is greater than current top-k results. The candidate can be skipped if this is true, otherwise we compute the final similarity. This process is often referred to as *early abandoning*.

Upper Bounding Similarity. For LORS, after accessing EdgII for all of the query edges, we know the list of common edges $Q \cap T$ in the scanned trajectory T. Therefore, the upper similarity bound is the sum of the travel length $|e|$ of all of these edges e, which can be easily computed.

$$\hat{S}^{\uparrow}_{\text{LORS}}(Q, T) = \sum_{e \in Q \cap T} |e| \tag{5}$$

Bound Reordering. To filter candidates that cannot be the top-k results, we sort all of the candidates by their upper bounds. With the sorted list of candidates, we can refine the candidate trajectory list from the beginning by computing the final similarity, and update the top-k result heap until the next candidate's upper bound is smaller than the k·th result's similarity.

In addition to efficient pruning based on LEVI and the upper bound computations, the real distance computation for LORS can be done without accessing the trajectory data because the filtering

procedure has already read the common edges overlapped with the query and the orderings of the whole trajectory T from LEVI, as it forms a sub-trajectory T_p of T. This results in the following lemma.

Lemma 1. $\forall T_p \in \text{can}, \hat{S}_{LORS}(Q, T_p) = \hat{S}_{LORS}(Q, T)$.

Proof. This lemma can be easily derived from Equation 4. Any edge that does not intersect with the query, denoted as $T - T_p$, cannot affect the similarity as the length of an edge will be added only when two edges overlap. □

Based on this Lemma, we can compute the final distance using a partial trajectory T_p, and the whole search procedure does not need to access the trajectory data. This can greatly improve efficiency in practice and reduce the space of data structures maintaining in the main memory.

6.3 Pruning with Existing Similarity Measures

A simple baseline to find the top-k trajectories for point-based similarity measures is to compute the similarity for every trajectory in D. For LCSS and EDR, we perform a single range query on a Grid-index to filter out the unrelated vertices [25]. For real distance functions such as DTW, Hausdorff and Fréchet distance, no trajectory can be filtered using VerII alone. In order to support dynamic pruning, we access all of the query vertices to scan new candidates based on the incremental range query presented in Section 5.3.

Incremental Vertex Scanning. For every query vertex, we access the nearby cells with vertices to produce a list of new candidate trajectories from the inverted lists of each vertex, which can in turn be completed using an incremental range search $IRS(q, i, g)$. The top-k result heap is updated with the candidates with the highest upper bounds as they are more likely to be the final results.

Unseen Upper Bound. The computation of the upper bound for unseen trajectories varies from one similarity measure to another. After i_{IRS} rounds of incremental range search over VGI for new candidate trajectories, the upper bound similarity for unseen trajectories is computed as:

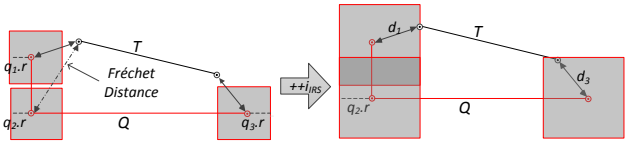
$$\hat{S}^{\uparrow}(Q, i_{IRS}) = \begin{cases} -\sum_{j=1}^{|Q|} q_j.r, \hat{S} = \text{DTW} \\ -\max_{j=1}^{|Q|} q_j.r, \hat{S} = \textit{Hausdorff}, \textit{Fréchet} \end{cases} \quad (6)$$

where $q_j.r$ is the scanning radius of point q_j, and can be computed as $q_j.r = r_{min} + i_{IRS} \times g$, and r_{min} is the vertical distance from q_j to the nearest edge of the cell c where q_k locates, as shown in Figure 3.

Lemma 2. $\forall T \in D - \textit{can}, \hat{S}(Q, T) < \hat{S}^{\uparrow}(Q, i_{\text{IRS}})$.

Proof. For a trajectory T that is not scanned by the incremental range search with a radius r ($T \in D - \textit{can}$), the distance from every point in T to the closest point in Q is greater than r. As the real distance function is pointwise, the overall similarity must be smaller than the sum of the maximum contributions from the query. □

Example 3. *In Figure 5(a), T is an unseen trajectory for Q w.r.t. three range queries (grey rectangles) – no point of T is inside. Each point in Q matches the nearest point in T without breaking the order constraint, and each match with an unseen trajectory such as T will have a distance greater than the scanning radius. Then, for any trajectory T which has not been scanned yet, we can estimate the upper bound, $\hat{S}^{\uparrow}(Q, i_{\text{IRS}}) = -(q_1.r + q_2.r + q_3.r)$ for DTW.*



(a) Upper bound of unseen trajectories (b) Upper bound of T

Figure 5: An example of the upper bound computation for similarity measures based on real distance, where DTW is composed of three two-way arrows while Fréchet distance is a single arrow with the maximum distance.

Table 1: Statistics of trajectory and road network datasets

	Porto	T-drive
#trajectories	1,652,742	250,997
#total points	80,635,629	11,757,455
#points per trajectory	49	47
#sampling rate (m)	113	660
average travel length (m)	5,632	31,056
Space (MB) of D	1,853	752
#Edges	150,761	126,827
#Vertices	114,099	54,198
Average edge length (m)	119	217
Space (MB) of G	11	5

The upper bound similarity $\hat{S}^{\uparrow}(Q, T)$ for a single trajectory T when it is scanned by a large range w.r.t. a query Q is the main optimization used to reduce computation of the full similarity for existing similarity measures, which is similar to computing the bound with linear complexity in Equation 5. For example in Figure 5(b), when T is scanned by point q_1 and q_3, we can compute the distance d_1 and d_3, for the point q_2 which has not scanned with T, its distance to T will be greater than $q_2.r$, then $\hat{S}^{\uparrow}(Q, T) = -(d_1 + q_2.r + d_3)$. There are many studies on how to tighten the bound computations for DTW [30], LCSS [39], EDR [3], Hausdorff [27] and Fréchet distance [1], and we can employ them for the corresponding measures in Algorithm 1. Details are omitted due to space limitations.

7 EXPERIMENTS

7.1 Experimental Setup

Datasets. We use the taxi trajectory datasets D of Porto[6] and Beijing T-drive [51]. The road network dataset G of each city is obtained from OpenStreetMap.[7] Table 1 describes the statistics of the trajectory datasets and road network dataset. The sampling rate here means the average distance d between two neighbor points in a trajectory (sampling a point every d meters.) As compared to T-drive, Porto has more trajectories, and the sampling rate is higher.

Implementation. We extend the map matching library of GraphHopper[8] with the shortest path-based optimization [29] to improve the efficiency of map-matching. We use FastPFor[9] to compress LEVI. All raw trajectories are mapped to the road network off-line at index time, and the total mapping time is shown in Table 2. All

[6] http://www.geolink.pt/ecmlpkdd2015-challenge/dataset.html
[7] https://www.openstreetmap.org/
[8] https://github.com/graphhopper/map-matching
[9] https://github.com/lemire/JavaFastPFOR

Table 2: Time spent on map matching, index construction and compression.

		Mapping	Index building	Compression
Porto	LEVI	3,840s	283 s	59s
	R-tree	–	245s	–
T-drive	LEVI	542s	214s	40s
	R-tree	–	66s	–

experiments were performed on a server using an Intel Xeon E5 CPU with 256 GB RAM running RHEL v6.3 Linux, implemented in Java. The index and data were saved to disk after construction, and were memory-mapped to perform the query processing. Before conducting the queries we ensure that LEVI is fully loaded in main memory.

Experimental Goals. The effectiveness and efficiency of Torch are evaluated in the next three subsections. We verify:
1) Index Performance – whether the compression techniques can significantly reduce the storage cost.
2) Efficiency of Search – whether LEVI and our pruning algorithm answers LORS similarity queries efficiently, as well as improve the performance of existing similarity measures over raw trajectories.
3) Effectiveness of LORS – whether LORS similarity is robust to the sampling rate, GPS error and point shifting.

7.2 Index Performance

Construction. Table 2 shows the time spent on index construction over the two datasets, including the time spent on the map-matching of the whole dataset, building and compressing LEVI. The "–" in the table means there is no map matching or compression when using an R-tree to index. Compared with indexing using an R-tree [14] on the raw trajectory directly, we spend most of the time on the map-matching. However, this is done off-line, and can further reduce the time when answering on-line queries.

Compressibility. Table 3 shows four aspects for each dataset: 1) the space of the raw trajectory dataset D and the R-tree (**Raw**); 2) the space of the VerII and EdgII, positional list, trajectory data, location table of edge and vertex on mapped trajectory data (**Map**), 3) the list compression (**ListCom**) based on PFor and VByte [20], 4) the trajectory reordering to compress the inverted lists (**Reorder**). The "–" in row "Raw" means that the index or similarity measure is not supported, and all other "–" mean that the index can not be compressed (ListCom and Reorder).

Observation 1. *For raw trajectories, the R-tree needs to create MBRs to bound points, requiring a significant amount of additional space. For example, the R-tree for Porto needs more than 2GB. When compared against indexing all of the points in trajectories using an R-tree, our index with the compression on lists and trajectory id reordering can significantly reduce the space. In particular, list compression and reordering can achieve a compression ratio of 15.1% – $764MB \rightarrow 281MB \rightarrow 116MB$ on the Porto dataset. This compression ratio makes it possible to process search queries over billions of trajectories. The right side of Table 3 shows the minimal storage to answer queries using each similarity measure. For example, LORS depends only on EdgII and the Edge table, and the storage of LORS is at most* $116 + 108 + 5 = 229MB$ *(in bold). When comparing the two datasets, we observe that the compression is better in the larger Porto dataset. For T-drive, map-matching needs to add more points to achieve competitive precision when compared with the raw trajectories. Overall, LORS achieves the most efficient search with the least footprint, needing only* $0.2GB$ *for Porto, while all other models require at least* $0.5GB$.

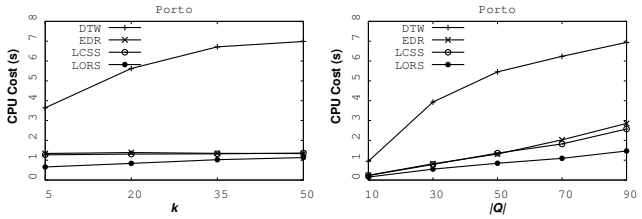


Figure 6: TSS with LEVI using different measures on Porto.

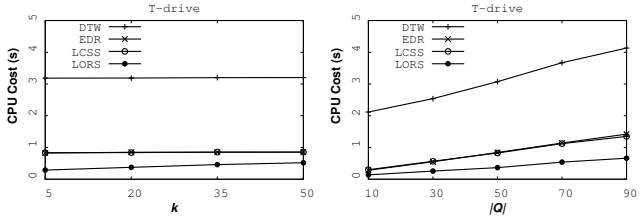


Figure 7: TSS with LEVI using different measures on T-drive.

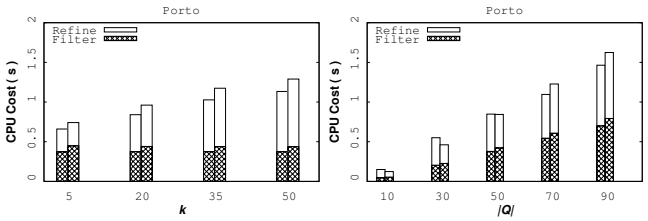


Figure 8: LORS with LEVI and Compressed LEVI on Porto.

7.3 Efficiency Evaluation

Comparison. Similarity search TSS and Boolean search BTS are both evaluated in this section. For TSS, we compare LORS with LCSS, and EDR/ DTW over the mapped and raw trajectories using the same query set. The Hausdorff and Fréchet distance are not shown here as they are much slower than the other four tested measures. To simulate the real queries, we generate a query pool of $1,000$ trajectories with a length $|Q|$ of 90 (the average length of trajectories in D) by randomly choosing from D, and choosing the $|Q|$ front edges incrementally in order to test the effect of parameter $|Q|$. For each query, we run it 5 times and report the average running time.

Similarity Search. To observe the effect of k and number of edges $|Q|$, we increase $k = 1, 5, \underline{20}, 35, 50$, $|Q| = 10, 30, \underline{50}, 70, 90$ to observe the performance by executing the queries in the query pool, where the underlined number is the default value.

Observation 2. *From Figure 6, we observe that LORS is always the most efficient when k and $|Q|$ are increased.*

Next, we explore the search time compared with state-of-the-art using raw trajectories, and the results on Porto are in shown in Figures 8,10,11,12. In each Figure, each column compares the indexes answering the same similarity measure, and it is composed

Table 3: Space occupation (MB) of index, trajectory and the minimum space to answer each of the six similarity measures. For VerII and EdgII, the left column shows the space of vertex id list, and the right column shows the space of the positional list. A breakdown of the index size under each similarity measure is as follows. LORS:{1, 6}; LCSS and EDR:{2, 3, 4, 5}; DTW, Hausdorff (Haus) and Fréchet: {2, 3, 5, 7} for raw data based on R-tree; the mapped data based on Grid-index is VGI {2, 3, 4, 5}.

		EdgII[1]		VerII[2]		VGI[3]	D^4	Vertex[5]	Edge[6]	R-tree[7]	LORS	LCSS	EDR	DTW	Haus	Fréchet
Porto	Raw	–	–	–	–	3,178	1,853	–	–	2,982	–	5,031	5,031	4,843	4,843	4,843
	Map	764	307	736	296	14	1,130	6	**5**		1,076	2,182	2,182	4,020	4,020	4,020
	ListCom	281	**108**	299	104	6	301	–	–		394	716	716	3,691	3,691	3,691
	Reorder	**116**	–	132	–	–	–	–	–		**229**	549	549	3,524	3,524	3,524
T-drive	Raw	–	–	–	–	741	752	–	–	410	–	1,493	1,493	1,162	1,162	1162
	Map	373	208	374	209	6	629	3	3		584	1,221	1,221	1,625	1,625	1,625
	ListCom	178	91	164	91	2	155	–	–		272	415	415	823	823	823
	Reorder	146	–	131	–	–	–	–	–		237	382	382	790	790	790

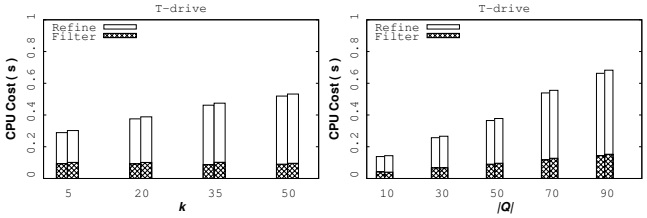


Figure 9: LORS with LEVI and Compressed LEVI on T-drive.

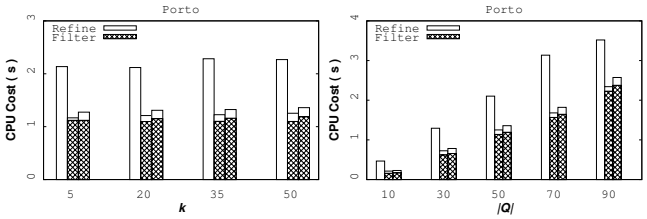


Figure 10: LCSS with Grid-index, LEVI, Compressed LEVI.

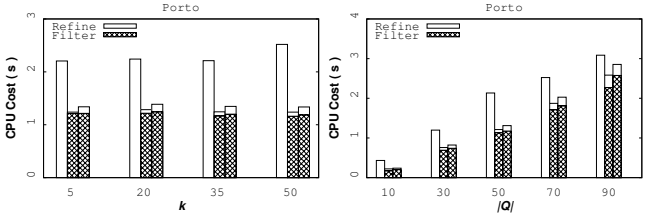


Figure 11: EDR with Grid-index, LEVI, Compressed LEVI.

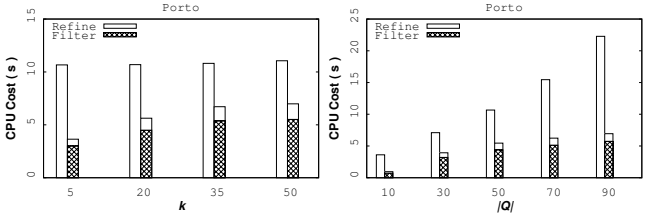


Figure 12: DTW with R-tree and LEVI.

of the time for filtering and refinement. For example, Figure 8 compares the search efficiency of LORS based on LEVI and compressed LEVI. Table 4 shows a further breakdown of the run time to the mapping, filtering, refinement, as well as the number of candidates after filtering, and refined candidates on the Porto dataset.

Observation 3. *First, LORS supports the search in the most efficient manner, followed by EDR and LCSS. Second DTW is improved the most by LEVI (up to 3 times). Third, the search over compressed data will degrade the efficiency by 10% to 20%, because of decompressing the compressed posting list to access the list of vertex ids. Last, from Table 4, it can be observed that filtering occupies the largest portion of run time in LORS.*

Table 4: Running time (s) and candidates break down of similarity search when $k = 20$, $|Q| = 50$, where "#can" means the number of candidates after filtering, "#refine" means the number of trajectories whose real distances are computed.

		Map	Filter	Refine	#can	#refine
Porto	LORS	0.002	0.378	0.469	228,848	44,641
	LCSS		1.187	0.171	286,218	13,935
	EDR		1.185	0.151	286,218	13,257
	DTW		5.512	1.475	288,820	134,236

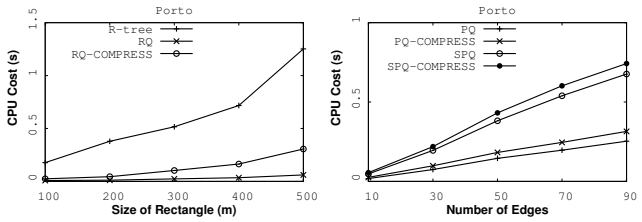


Figure 13: BTS: range query (left) and path query (right).

Boolean Search. Given a rectangular region, a range query (RQ) searches all the trajectories that cross this region. We randomly generate 1,000 points in the space of each city, and further produce the rectangles centered at the point with a side length of $r = 100m, 200m, 300m, 400m, 500m$. Then we increase the size of rectangle to observe the performance. Based on the query pool with 1,000 trajectories used in the similarity search, we increase the number of query edges on the former query to observe the performance of path query (PQ) and strict path query (SPQ), i.e., $|Q| = 10, 30, 50, 70, 90$.

Observation 4. *We compare our performance with range queries over an R-tree of raw trajectories. In Figure 13, a range query with LEVI is more efficient than with an R-tree [16] or a Grid-index [25], with up to a ten fold improvement. A path query (PQ) is more efficient than a strict path query (SPQ), as an SPQ imposes ordering constraints. PQ and SPQ with as many as 90 edges can still be answered within 1*

Table 5: The robustness evaluation to sampling rate, GPS error and point shifting.

		Sampling Rate						GPS Error			Point Shifting		
		LORS	LCSS	EDR	DTW	Fréchet	Haus	LORS	LCSS	EDR	LORS	LCSS	EDR
Porto	P@5	0.959	0.825	0.872	0.887	0.857	0.841	0.987	0.600	0.587	0.969	0.793	0.843
	P@10	0.941	0.793	0.839	0.840	0.830	0.809	0.969	0.550	0.529	0.964	0.791	0.840
	P@15	0.921	0.778	0.819	0.813	0.815	0.788	0.946	0.532	0.506	0.959	0.790	0.831
	NDCG@5	0.966	0.846	0.888	0.902	0.869	0.858	0.991	0.641	0.630	0.972	0.805	0.851
	NDCG@10	0.952	0.819	0.861	0.868	0.849	0.832	0.979	0.595	0.579	0.968	0.801	0.847
	NDCG@15	0.938	0.804	0.845	0.846	0.836	0.815	0.962	0.575	0.555	0.964	0.794	0.839
T-drive	P@5	0.982	0.895	0.816	0.981	0.950	0.909	0.999	0.880	0.760	0.958	0.881	0.801
	P@10	0.974	0.916	0.824	0.969	0.958	0.916	0.988	0.868	0.720	0.953	0.914	0.829
	P@15	0.964	0.912	0.831	0.933	0.962	0.922	0.976	0.836	0.681	0.952	0.927	0.857
	NDCG@5	0.984	0.882	0.823	0.981	0.948	0.907	0.998	0.877	0.775	0.960	0.864	0.807
	NDCG@10	0.978	0.900	0.825	0.974	0.954	0.912	0.990	0.871	0.745	0.956	0.891	0.822
	NDCG@15	0.972	0.901	0.829	0.951	0.958	0.916	0.982	0.851	0.716	0.955	0.904	0.841

second, which should fulfill the goal of near real-time traffic monitoring. PQ and SPQ with compression can be slower, but still 5 *times faster than an R-tree based range query.*

7.4 Effectiveness Evaluation

To the best of our knowledge, no ground truth relevance labels exist for trajectory search over road networks, which could be used to evaluate the robustness of different similarity measures. Torch employs a relevance judgment method where the ground truth is generated by creating trajectories on a given path in the road network, and checking whether they can be still retrieved as a top-k result. Such a path-based ground truth generation is also widely used in map matching [26].

Ground Truth Simulation. We randomly choose 2,000 trajectories from each dataset, and map them to the path in the road network. For each path, we generate k trajectories from this path using a GPS simulator [48], and inject them into the raw trajectory dataset D. To simulate the GPS sampling of a path, we mainly consider three features: sampling rate, GPS error, and point shifting. In particular, we maintain three ground truth sets by changing the sampling rate, injecting GPS error and conducting point shifting.

For the sampling rate, we take each query trajectory as the base, and add 1, 2, ..., k points into the edge between two adjacent vertices to generate k trajectories successively. For the GPS error, we re-sample each point and move it by a distance of 2m to 7.8m as the GPS has a global average user range error (URE) of $\leq 7.8m$.[10] For point shifting, we shift all of the points of the raw query trajectory along the road segments by 1m, 2m, ..., $\frac{k}{2}$ in two directions. For all the search results of a query, we judge them with three grades: 1) ***2*** for those trajectories in the generated ground truth of query. 2) ***1*** for those trajectories which are not in the ground truth but overlap with the query. 3) ***0*** for all remaining trajectories in D. To guarantee that all of the trajectories in D are labeled with only one of the three grades, we remove all of the trajectories that overlap with the queries before injecting the ground truth set.

Comparison. We compare LORS with the other five similarity measures including DTW, LCSS, EDR, Hausdorff and Fréchet distance over the raw trajectories, as all the state-art-of methods on trajectory similarity search previously used raw trajectories [38, 47]. The comparison is extensively conducted by changing $k = 5, 10, 15$ using three simulated ground-truth sets. We employ two common metrics to measure the precision, they are top-k result *precision* (**P@**k) and *normalized discounted cumulative gain* (**NDCG@**k) [15].

Observation 5. *We compare all six similarity measures by sampling rate, GPS error and point shifting, as shown in Table 5. We observe that* LORS *consistently has the highest search precision. The main reason for non-relevant results is that the raw trajectory is not mapped to the original route, and may lead to mismatches with the query. As* k *increases, the precision of all measures degrades as the results contain more trajectories which are not in the ground truth set. For GPS errors and point shifting, we choose the two most efficient measures* LCSS *and* EDR *to compare due to space limitations. We observe that* LORS *is robust to GPS errors and point shifting, while* LCSS *and* EDR *have low precision as they used a threshold to match points, and the two original matching points are not matched as the GPS error and point shifting divide them. Interestingly, T-drive has a higher precision than Porto for each similarity measure, especially* DTW*. This is because T-drive has fewer trajectories, and the ground truth queries injected are therefore easier to identify in the top-k results. However,* DTW *spends more than 5 times longer than* LORS *to complete the search. To summarize, among all six measures,* LORS *is the most robust to sampling rate, GPS error and point shifting.*

7.5 Further Discussion

By conducting extensive experiments on the efficiency and effectiveness of Torch, we show that LORS, driven by LEVI, has the best performance among existing similarity models. By aggregating the above observations, we find that: 1) LORS effectively solves the problem of sensitivity to sampling rate, and it is also robust to GPS error and point shifting. 2) LEVI improves the efficiency of existing similarity measures, sometimes significantly. 3) Data is more easily reduced using common compression techniques from the IR domain, and the index is adaptive. We only need to load a subset of index components into main memory to answer a query.

[10] https://www.gps.gov/systems/gps/performance/accuracy/

8 CONCLUSIONS

This paper proposes a trajectory search engine called Torch to efficiently answer two common types queries (Boolean search and top-k similarity search) over trajectories on road networks. In Torch, the three-level design and applications of inverted index, compression and effectiveness evaluation from information retrieval simplify the trajectory storage and retrieval, enable a new similarity model called LORS based on road segments, and accelerate the search based on the highly compressible LEVI. In the future, we would like to further enrich the supported queries, such as the time-sensitive queries, and implement advanced analytic operators including trajectory clustering and prediction on top of Torch.

ACKNOWLEDGMENTS

This work was partially supported by ARC DP170102726, DP180102050, DP170102231, and NSFC 61728204, 91646204. Zhifeng Bao is a recipient of Google Faculty Award.

REFERENCES

[1] Pankaj K. Agarwal, Rinat Ben Avraham, Haim Kaplan, and Micha Sharir. 2014. Computing the discrete fréchet distance in subquadratic time. *SIAM J. Comput* 43, 2 (2014), 429–449.
[2] Jie Bao, Ruiyuan Li, Xiuwen Yi, and Yu Zheng. 2016. Managing massive trajectories on the cloud. In *SIGSPATIAL*. 41–50.
[3] Lei Chen, M. Tamer Özsu, and Vincent Oria. 2005. Robust and fast similarity search for moving object trajectories. In *SIGMOD*. 491–502.
[4] Yanping Chen, Eamonn Keogh, Bing Hu, Nurjahan Begum, Anthony Bagnall, Abdullah Mueen, and Gustavo Batista. 2015. The UCR Time Series Classification Archive. (2015).
[5] Zaiben Chen, Heng Tao Shen, Xiaofang Zhou, Yu Zheng, and Xing Xie. 2010. Searching trajectories by locations-an efficiency study. In *SIGMOD*. 255–266.
[6] W. Bruce Croft, Donald Metzler, and Trevor Strohman. 2009. *Search Engines - Information Retrieval in Practice*. Pearson Education.
[7] Philippe Cudre-Mauroux, Eugene Wu, and Samuel Madden. 2010. TrajStore: an adaptive storage system for very large trajectory data sets. In *ICDE*. 109–120.
[8] J. Shane Culpepper and Alistair Moffat. 2006. Phrase-based searching in compressed text. In *SPIRE*. 337–345.
[9] J. Shane Culpepper and Alistair Moffat. 2010. Efficient set intersection for inverted indexing. *ACM Transactions on Information Systems* 29, 1 (2010), 1–25.
[10] Victor Teixeira de Almeida and Ralf Hartmut Güting. 2005. Indexing the trajectories of moving objects in networks. *GeoInformatica* 9, 1 (2005), 33–60.
[11] Laxman Dhulipala, Igor Kabiljo, Brian Karrer, Giuseppe Ottaviano, Sergey Pupyrev, and Alon Shalita. 2016. Compressing graphs and indexes with recursive graph bisection. In *SIGKDD*. 1535–1544.
[12] Aiden Roger Doherty, Cathal Gurrin, Gareth J F Jones, and Alan F Smeaton. 2006. Retrieval of similar travel routes using GPS tracklog place names. In *GIR@SIGIR*. 2–5.
[13] Ronald Fagin, Amnon Lotem, and Moni Naor. 2003. Optimal aggregation algorithms for middleware. *J. Comput. System Sci.* 66, 4 (2003), 614–656.
[14] Antonin Guttman. 1984. R-trees: a dynamic index structure for spatial searching. *ACM SIGMOD Record* 14, 2 (1984), 47–57.
[15] Kalervo Järvelin and Jaana Kekäläinen. 2002. Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems* 20, 4 (2002), 422–446.
[16] Eamonn Keogh. 2002. Exact indexing of dynamic time warping. In *VLDB*. 406–417.
[17] Kyoung-Sook Kim, Koji Zettsu, Yutaka Kidawara, and Yasushi Kiyoki. 2008. Path query processing for moving objects on road networks. In *MobIR@SIGIR*. 32–39.
[18] Satoshi Koide, Yukihiro Tadokoro, and Takayoshi Yoshimura. 2015. SNT-index: Spatio-temporal index for vehicular trajectories on a road network based on substring matching. In *UrbanGIS@SIGSPATIAL*. 1–8.
[19] Benjamin Krogh and Christian S Jensen. 2016. Efficient in-memory indexing of network-constrained trajectories. In *SIGSPATIAL*. 17:1–17:10.
[20] Daniel Lemire and Leonid Boytsov. 2015. Decoding billions of integers per second through vectorization. *Software: Practice and Experience* 45, 1 (2015), 1–29.
[21] Yanhua Li, Chi-yin Chow, Ke Deng, and Mingxuan Yuan. 2015. Sampling big trajectory data. In *CIKM*. 941–950.
[22] Yin Lou, Chengyang Zhang, Yu Zheng, Xing Xie, Wei Wang, and Yan Huang. 2009. Map-matching for low-sampling-rate GPS trajectories. In *SIGSPATIAL*. 352–361.
[23] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. 2008. *An Introduction to Information Retrieval*. Cambridge University Press.
[24] Alistair Moffat and J. Shane Culpepper. 2007. Hybrid bitvector index compression. In *ADCS*. 25–31.
[25] Michael D Morse and Jignesh M Patel. 2007. An efficient and accurate method for evaluating time series similarity. In *SIGMOD*. 569–580.
[26] Paul Newson and John Krumm. 2009. Hidden Markov map matching through noise and sparseness. In *SIGSPATIAL*. 336–343.
[27] Sarana Nutanong, Edwin H. Jacox, and Hanan Samet. 2011. An incremental Hausdorff distance calculation algorithm. *PVLDB* 4, 8 (2011), 506–517.
[28] Dieter Pfoser, Christian S. Jensen, and Yannis Theodoridis. 2000. Novel approaches to the indexing of moving object trajectories. In *VLDB*. 395–406.
[29] Mohammed Quddus and Simon Washington. 2015. Shortest path and vehicle trajectory aided map-matching for low frequency GPS data. *Transportation Research Part C: Emerging Technologies* 55 (2015), 328–339.
[30] Thanawin Rakthanmanon, Bilson Campana, Abdullah Mueen, Gustavo Batista, Brandon Westover, Qiang Zhu, Jesin Zakaria, and Eamonn Keogh. 2012. Searching and mining trillions of time series subsequences under dynamic time warping. In *SIGKDD*. 262–270.
[31] Sayan Ranu, P Deepak, Aditya D Telang, Prasad Deshpande, and Sriram Raghavan. 2015. Indexing and matching trajectories under inconsistent sampling rates. In *ICDE*. 999–1010.
[32] Gook-pil Roh, Jong-won Roh, Seung-won Hwang, and Byoung-kee Yi. 2011. Supporting pattern-matching queries over trajectories on road networks. *IEEE Transactions on Knowledge and Data Engineering* 23, 11 (2011), 1753–1758.
[33] Iulian Sandu Popa, Karine Zeitouni, Vincent Oria, Dominique Barth, and Sandrine Vial. 2011. Indexing in-network trajectory flows. *VLDB Journal* 20, 5 (2011), 643–669.
[34] Falk Scholer, Hugh E. Williams, John Yiannis, and Justin Zobel. 2002. Compression of inverted indexes for fast query evaluation. In *SIGIR*. 222–229.
[35] Renchu Song, Weiwei Sun, Baihua Zheng, and Yu Zheng. 2014. PRESS: a novel framework of trajectory compression in road networks. *PVLDB* 7, 9 (2014), 661–672.
[36] Eleftherios Tiakas, Apostolos Papadopoulos, Alexandros Yannis Manolopoulos Nanopoulos, Dragan Stojanovic, and Slobodanka Djordjevic-Kajan. 2009. Searching for similar trajectories in spatial networks. *The Journal of Systems & Software* 82, 5 (2009), 772–788.
[37] Howard Turtle and James Flood. 1995. Query evaluation: strategies and optimizations. *Inf. Process. Manage.* 31, 6 (1995), 831–850.
[38] Michail Vlachos, Marios Hadjieleftheriou, Dimitrios Gunopulos, and Eamonn Keogh. 2003. Indexing multi-dimensional time-series with support for multiple distance measures. In *SIGKDD*. 216–225.
[39] Michail Vlachos, George Kollios, and Dimitrios Gunopulos. 2002. Discovering similar multidimensional trajectories. In *ICDE*. 673–684.
[40] Haozhou Wang, Han Su, Kai Zheng, Shazia Sadiq, and Xiaofang Zhou. 2013. An effectiveness study on trajectory similarity measures. In *ADC*. 13–22.
[41] Haozhou Wang, Kai Zheng, Jiajie Xu, Bolong Zheng, and Xiaofang Zhou. 2014. SharkDB: an in-memory column-oriented trajectory storage. In *CIKM*. 1409–1418.
[42] Sheng Wang, Zhifeng Bao, J Shane Culpepper, Timos Sellis, and Gao Cong. 2018. Reverse k nearest neighbor search over trajectory. *IEEE Transactions on Knowledge and Data Engineering* 30, 4 (2018), 757 – 771.
[43] Sheng Wang, Zhifeng Bao, J Shane Culpepper, Timos Sellis, Mark Sanderson, and Xiaolin Qin. 2017. Answering top-k exemplar trajectory queries. In *ICDE*. 597–608.
[44] Sheng Wang, Zhifeng Bao, Shixun Huang, and Rui Zhang. 2018. A unified processing paradigm for interactive location-based web search. In *WSDM*. 601–609.
[45] Xiaoyue Wang, Abdullah Mueen, Hui Ding, Goce Trajcevski, Peter Scheuermann, and Eamonn Keogh. 2013. Experimental comparison of representation methods and distance measures for time series data. *Data Mining and Knowledge Discovery* 26, 2 (2013), 275–309.
[46] Jung-im Won, Sang-wook Kim, Ji-haeng Baek, and Junghoon Lee. 2009. Trajectory clustering in road network environment. In *CIDM*. 299–305.
[47] Dong Xie, Feifei Li, and Jeff M Phillips. 2017. Distributed trajectory similarity search. *PVLDB* 10, 11 (2017), 1478–1489.
[48] Jianqiu Xu and Ralf Hartmut G. 2012. MWGen : a mini world generator. In *MDM*. 258–267.
[49] Hao Yan, Shuai Ding, and Torsten Suel. 2009. Compressing term positions in web indexes. In *SIGIR*. 147–154.
[50] Byoung-Kee Yi, H V Jagadish, and Christos Faloutsos. 1998. Efficient retrieval of similar time sequences under time warping. In *ICDE*. 201–208.
[51] Jing Yuan, Yu Zheng, Chengyang Zhang, Wenlei Xie, Xing Xie, Guangzhong Sun, and Yan Huang. 2010. T-drive : driving directions based on taxi trajectories. In *SIGSPATIAL*. 99–108.
[52] Yu Zheng. 2015. Trajectory data mining : an overview. *ACM Trans. Intell. Syst. Technol.* 6, 3 (2015), 1–41.