

# Fast Large-Scale Trajectory Clustering (Technical Report)

Sheng Wang  
RMIT University

sheng.wang@rmit.edu.au zhifeng.bao@rmit.edu.au shane.culpepper@rmit.edu.au

Timos Sellis  
Swinburne University of  
Technology  
tsellis@swin.edu.au

Zhifeng Bao  
RMIT University

J. Shane Culpepper  
RMIT University

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

## ABSTRACT

In this paper, we study the problem of large-scale trajectory data clustering called  $k$ -paths, where we aim to efficiently identify  $k$  “representative” road network paths. It benefits a range of applications such as traffic monitoring, public transit planning, and site selection. By combining map matching with an efficient intermediate representation of trajectories, we are able to solve  $k$ -paths using an efficient *edge-based distance* (EBD) measure, and an efficient clustering algorithm. Experiments verify that we can cluster millions of taxi trajectories in around one minute, achieving improvements of up to two orders of magnitude over state-of-the-art solutions for the trajectory clustering problem.

### Reference Format:

Sheng Wang, Zhifeng Bao, J. Shane Culpepper, Timos Sellis, Xiaolin Qin. Fast Large-Scale Trajectory Clustering (Technical Report). xxxx, xx(xxx): xxxx-yyyy, 2019.  
DOI:

## 1. INTRODUCTION

Ubiquitous trajectory data is being generated from a diverse range of resources, such as Global Positioning System (GPS) devices, cameras, and radio frequency identification (RFID) readers [73]. Trajectory data volume is huge – a car carrying embedded mobile broadband chips can produce up to 25 gigabytes of cloud-based data in an hour, including GPS routes [1].

In this paper, we study a fundamental problem, the clustering of large-scale vehicle trajectories. Despite a great deal of progress on this problem since 2007 [46, 39] (summarized in Table 1), significant challenges remain in designing scalable clustering algorithms for large-scale trajectory data. Moreover, many clustering algorithms require users to make difficult parameter choices, such as density thresholds [46, 47], in order to generate candidate line segments. Tuning

these sensitive parameters itself is challenging for even domain experts (due to different degrees of limited domain knowledge on the data) [32, 71].

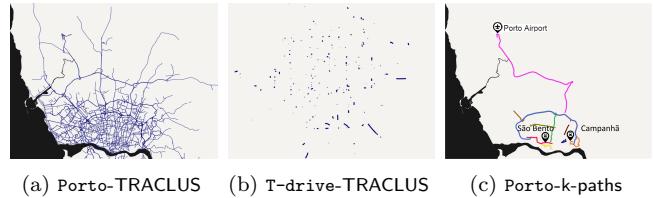


Figure 1: A glimpse of TRACCLUS and  $k$ -paths.

Figure 1(a)(b) visualize the results of the widely used trajectory clustering algorithm TRACCLUS [46] on two taxi trip datasets: **Porto** [6] and **T-drive** [80]. TRACCLUS uses a density threshold to group all frequent edges into the final clusters (the threshold is set as the average edge frequency). In **Porto**, the skeleton of the road network is identified by TRACCLUS, but it is hard to observe any discernible trends; in **T-drive** only discrete edges which appear disconnected are identified as the clustered result. The above indicates that the threshold selection is highly sensitive to the data.

In this paper, we propose  $k$ -paths, which aims to cluster trajectories into  $k$  groups where  $k$  *representative real paths* are selected as the delegates, as shown in Figure 1(c) where  $k = 10$ .  $k$ -paths is reminiscent of the classical  $k$ -means [53], where in both problems  $k$  is the only parameter required from user.  $k$ -paths is useful in many upstream applications, such as:

- **Scenario 1: Traffic Flow Analysis.** A traffic analyst needs to find the  $k$  frequently travelled paths [52] to visually analyze complex transportation networks [8, 27].
- **Scenario 2: Bus Route Planing.** A transportation department wants to open  $k$  new bus routes to meet growing demands [35, 43] using historical vehicle trip records [80].
- **Scenario 3: Facility Site Selection.** A company plans to open  $k$  petrol stations around the busiest routes in a city. This problem is commonly referred to as the trajectory-driven facility deployment problem [81, 48].

Efficient  $k$ -means with Lloyd’s algorithm [50] has previously been investigated for large-scale point data [23, 56]. By choosing  $k$  objects as the initial centroids, it iteratively assigns each object to the nearest centroid, and refines the new centroid in each cluster.  $k$ -paths can be answered by extending Lloyd’s algorithm. However, scaling this approach is challenging since the assignment requires  $\mathcal{O}(nkt)$  distance

computations, where  $n$  is the number of trajectories and  $t$  is the number of iterations. Meanwhile, a simple refinement requires  $\mathcal{O}(n^2t)$  distance computations (see a thorough analysis in Section 3.3). Therefore, answering  $k$ -paths requires a prodigious number of distance computations, and existing trajectory distance measures are expensive to compute, e.g., Edit Distance on Real Sequences (EDR) [17] has quadratic complexity. When using EDR, our experiments showed that  $k$ -paths did not converge after several days when clustering as few as 10,000 trajectories.

Besides the computational costs, clustering quality is another challenge. Most existing work [46, 14] cluster the raw trajectories using data points sampled from GPS readings. In our recent work [69], it was found that raw trajectories have precision problems derived from GPS errors and sampling rate. We proposed a pre-processing technique that used *map-matching* [51] of raw trajectories to road network paths when constructing indexes, which to our best knowledge is the de facto for effective trajectory analysis in the road network [79]. This approach can significantly reduce both storage and computational complexity while achieving better precision when measuring the similarity between two trajectories. Similarly, we adopt map-matching before clustering in this work. However, the distance computation cost remains quadratic.

To overcome the above obstacles when scaling  $k$ -paths to larger collections, we propose an efficient clustering framework which has two important properties:

**1) A quasilinear distance measure.** We propose a new distance measure EBD, as an extension of the recently proposed distance measure LORS [69]. EBD computes the distance between two trajectories based on the intersecting road segments and travel length. It can reduce the distance computation cost from quadratic (in LORS) to quasilinear, and meanwhile return the same score when measuring the similarity between two trajectories, and allow compressed trajectory representations to be used during clustering.

**2) Fewer distance computations.** We design novel indexing techniques to significantly reduce the number of distance computations in the assignment and refinement phases of the clustering algorithm. After proving that EBD satisfies the triangle inequality (metric), we employ a lower bound technique to prune the computational space and propose an indexing framework to accelerate the clustering. To refine the centroid path more efficiently, we present a linear-time approach that exploits the *length histogram* and an *edge histogram*. We further extract the centroid path by traversing the road network graph, which is independent of the number of trajectories.

To summarize, we have made the following contributions:

- We define a fundamental trajectory clustering problem  $k$ -paths based on a map-matched trajectory modeling method (Section 3), combined with a novel distance measure EBD which is metric and supports efficient  $k$ -paths clustering (Section 4).
- We propose a clustering framework with low complexity for  $k$ -paths based on Lloyd’s algorithm, coupled with lower bounds based assignment and histograms based refinement on improving clustering performance (Section 5).
- We design an indexing framework called PIG to further prune during the assignment, and transform the refinement to a graph traversal problem—CPEP (Section 6).

- We evaluate the efficiency, scalability, and effectiveness of EBD-based  $k$ -paths by comparing with five widely-used distance measures, and the state-of-the-art trajectory clustering work [46, 14] using two real-world datasets. A case-study based on visualization verifies that the most frequent paths can be identified accurately (Section 7).

All proofs for conjectures and lemmas in this paper can be found in our technical report [68], and the source code is available<sup>1</sup> for reproducibility.

## 2. RELATED WORK

**Trajectory Modeling.** A set of coordinates denoted by two floats is a traditional representation of a trajectory for storage, search, and analytics. To further reduce space, converting the raw data to a vector of the same length is proposed [75]. Such a simple conversion is often inefficient when used for hashing or training machine learning models. Recently, storing raw trajectories as a path on a road network using *map matching* [51] has been shown to be more effective in terms of storage [65, 74] and retrieval performance [69, 45]. Wang et al. [69] modeled the road network as a directed graph, and each road is an edge with a unique ID, and a trajectory can be represented as a sequence of unsorted integers.

**Trajectory Distance Measures.** Many distance measures have been proposed for trajectory data. Common measures include Dynamic Time Warping (DTW) [41], Longest Common Sub-sequence (LCSS) [66], Edit Distance on Real sequence (EDR) [17], Hausdorff distance [57], Discrete Fréchet distance [25], and Edit distance with Real Penalty (ERP) [16]. However, all these existing measures based on points suffer from the high complexity (quadratic) and sensitivity to GPS errors, sampling rate, point shift, or parameter (EDR, LCSS, and ERP all need one parameter) when applied to raw trajectories from vehicles. In our recent work [69], experiments verified the robustness of a new distance measure: *the longest overlapped road segments* (LORS). Yuan et al. [79] further normalized LORS to measure the distance between two trajectories by considering the length of two trajectories.

Table 1: A summary of existing trajectory clustering work (“P” denotes “Partition”, “D” denotes “Density”).

Work	Type	Measure	Data Scale	#Para	Time
<b>k</b> -paths	P	EBD	Million (Vehicle)	1	Minute
[14]		Hausdorff	422 (Vehicle)	4	Hours
[75]		DTW	4700 (Vessels)	2	NA
[27]		Euclidean	372,601 (Vehicle)	2	2497s
[30]	P	Fréchet	2 soccer players	2	700s
[61]		ERP	1100 (Vehicle)	4	10s
[73]		EDR	100 (Cellular)	3	400s
[8]		-	176,000 (Vehicle)	$ D $	NA
[47]		-	7000 (Vehicle)	3	100s
[35]	D	Hausdorff	5000 (Vehicle)	6	120s
[46]		-	570 (Hurricane)	2	NA
[7]		Fréchet	20,000 (Vehicle)	3	50.4hrs

**Trajectory Clustering.** As shown in Table 1, existing trajectory clustering methods can be divided into two types [78], Partition-based [14, 27, 75, 61, 30, 37, 36] and Density-based [46, 35, 47, 7, 8, 31]. Note that partition-based clustering is popular for not only vehicle trajectory data, but also time series data such as  $k$ -Shape [58, 38]. Based on this categorization, three important observations about current techniques can be made: 1) Existing solutions are only

<sup>1</sup><https://github.com/tgbnhy/k-paths-clustering>

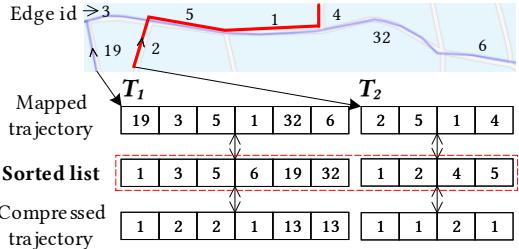


Figure 2: Trajectory data modeling and compression.

tractable for small datasets such as animals and hurricanes. For example, the most cited trajectory clustering solution [46] clusters only 570 trajectories (so we optimized [46] to generate Figure 1, see Section 7.3.1 for details); 2) Clustering is done on raw trajectory data composed of points, and the distance measure computations need quadratic complexity; 3) Most solutions have at least two threshold parameters which are highly sensitive to the dataset, making them difficult to reproduce or use in practice (thus the running time shown in Table 1 is reported from the original papers). In contrast, our solution for  $k$ -paths requires no threshold parameterization, is simpler to reproduce, and has a novel distance measure-EBD which is both scalable and effective.

### 3. DEFINITIONS & PRELIMINARY

#### 3.1 Trajectory Data Modeling

**DEFINITION 1. (Point)** A point  $p = \{lat, lng\}$  contains the latitude  $lat$ , the longitude  $lng$ .

**DEFINITION 2. (Raw Trajectory)** A trajectory  $T$  of travel length  $|T|$  is in the form of  $\{p_1, p_2, \dots, p_m\}$ , where each  $p_i$  is a point.

**DEFINITION 3. (Road Network)** A road network is a directed 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, each vertex has a unique id allocated from 1 to  $|V|$ .

**DEFINITION 4. (Path)** A path  $P$  is composed by a set of connected road edges  $e_1 \rightarrow e_2 \rightarrow \dots \rightarrow e_m$  in  $G$ . The travel length of  $P$  is defined as the sum of length of all edges.

**DEFINITION 5. (Map-Matched Trajectory)** Given a raw trajectory  $T$  and a road network  $G$ , we map  $T$  to a set of connected edges in  $G$ , such that  $T : e_1 \rightarrow e_2 \rightarrow \dots \rightarrow e_m$ .

In the rest of this paper, we use *trajectory* to represent the *mapped trajectory*. Frequent notations are summarized in Table 2. Example 1 shows a case of converting a raw trajectory to an *edge id list*.

**EXAMPLE 1.** As shown in Figure 2, two trajectories  $T_1$  (blue) and  $T_2$  (red) have been mapped into the road network. Note that we just show the one-way edges here, and label each of them with an integer. Then, we can store them as  $T_1 = \{19, 3, 5, 1, 32, 6\}$  and  $T_2 = \{2, 5, 1, 4\}$ .

#### 3.2 $k$ -paths Trajectory Clustering

**DEFINITION 6. ( $k$ -paths Trajectory Clustering)** Given a set of trajectories  $\{T_1, T_2, \dots, T_n\}$ ,  $k$ -paths aims to partition the  $n$  trajectories into  $k$  ( $k \leq n$ ) clusters  $S = \{S_1, S_2, \dots, S_k\}$  to minimize the objective function:

Table 2: Summary of notations.

Symbols	Description
$T_i, D$	A trajectory, and the dataset
$ T_i $	The travel length of trajectory $T_i$
$G, E, V, P$	The road network, edges, vertices and path
$O, S$	The objective function, clusters
$a'(i), a(i)$	$T_i$ 's previously and newly assigned cluster ids
$\mu_j, \mu_j'$	Cluster $S_j$ 's previous and current centroid paths
$ub(i)$	The upper bound distance from $T_i$ to its nearest cluster
$lb(i)$	The lower bound distance from $T_i$ to its second nearest cluster
$cd(j), cb(j)$	The centroid drift and bound of $\mu_j$
$EH, ALH$	The edge and accumulated length histograms
$\ e\ , \ G_j\ $	The weight of edge $e$ and frequency graph $G_j$

$$O = \arg \min_S \sum_{j=1}^k \sum_{T_i \in S_j} \text{Dist}(T_i, \mu_j) \quad (1)$$

where each cluster  $S_j$  has a centroid path  $\mu_j$  which should be a path in  $G$ , and  $\text{Dist}$  is the trajectory distance measure. The key differences between  $k$ -paths and  $k$ -means are three-fold: (1) trajectories can be of varying lengths instead of fixed-length vectors in a Euclidean space; (2) a trajectory distance measure  $\text{Dist}$  for two trajectories must be defined; (3) the centroid path  $\mu_j$  cannot be found by simply computing the mean value of all trajectories in the cluster. Similar to a variant of  $k$ -means called  $k$ -medoids [59], an existing trajectory can be chosen as the centroid path.

#### 3.3 Lloyd's Algorithm for $k$ -paths

Since  $k$ -paths is a direct variant of  $k$ -means to solve the trajectory clustering problem, the processing framework of the well-known Lloyd's algorithm [50] for  $k$ -means can be extended to solve  $k$ -paths, consisting of three steps:

- 1) **Initialization.** Randomly choose  $k$  trajectories from  $D$  as the initial centroid paths (seeds):  $\{\mu_1, \dots, \mu_k\}$ .
- 2) **Assignment.** Find the nearest centroid path  $\mu_j$  for every trajectory  $T_i$  in the database, and assign it to the centroid path's affiliated cluster, denoted as  $a(i) = j$ .<sup>2</sup>
- 3) **Refinement.** Update the centroid path of each cluster by choosing an existing trajectory that can minimize the sum of distance to all other trajectories in the cluster. If all the centroid paths stop changing, return the  $k$  centroid paths as the final result; otherwise, go to step 2).

There are two core challenges in the assignment and refinement steps when using Lloyd's algorithm to solve  $k$ -paths:

**Challenge 1.** The complexity of the assignment step is  $\mathcal{O}(nk) \times \mathcal{O}(\text{dis})$ , where  $n$  is the total number of trajectories,  $k$  is the number of clusters and  $\mathcal{O}(\text{dis})$  is the complexity of distance computation which is quadratic when using existing distance measures.

**Challenge 2.** The complexity of computing the mean in the refinement step is constant in  $k$ -means, but it does not hold in  $k$ -paths. Even though we can extend  $k$ -medoids [59] to choose an existing trajectory as a new centroid path, we need to build a distance matrix with a complexity of  $\mathcal{O}(n^2) \times \mathcal{O}(\text{dis})$ .

<sup>2</sup>In the rest of the paper, indices  $i$  and  $j$  always refer to trajectory and cluster indices, respectively.

## 4. A NOVEL DISTANCE MEASURE

Large-scale clustering requires a distance measure that is precise and cheap to compute. The distance measure LORS [69], which is the state-of-the-art in term of precision, can be extended for  $k$ -paths without sacrificing effectiveness.

### 4.1 Preliminary of LORS

LORS measures the similarity based on the length of overlapped edges and also ensures that matched edges do not violate ordering constraints, which is also known as *local time shifting* [17]. Formally, LORS is defined as:

$$\hat{S}(T_1, T_2) = \begin{cases} 0, & \text{if } T_1 \text{ or } T_2 \text{ is empty} \\ |e_{1m}| + \hat{S}(H(T_1), H(T_2)), & \text{if } e_{1m} = e_{2x} \\ \max(\hat{S}(H(T_1), T_2), \hat{S}(T_1, H(T_2))), & \text{otherwise} \end{cases} \quad (2)$$

where  $T_1 = (e_{11}, e_{12}, \dots, e_{1m})$  and  $T_2 = (e_{21}, e_{22}, \dots, e_{2x})$ ;  $|e_{1m}|$  is the travel length of graph edge  $e_{1m}$ .  $H(T_1) = (e_{11}, \dots, e_{1m-1})$  is the sub-trajectory of  $T_1$  minus the last edge  $e_{1m}$ . For example, to calculate LORS between  $T_1$  and  $T_2$  in Figure 2, we can use a dynamic programming approach similar to existing measures [17, 76], and return  $\hat{S}(T_1, T_2) = |e_5| + |e_1|$  as they share a common sub-trajectory  $(e_5, e_1)$ . The detailed computation using dynamic programming with a 2D matrix can be found in [68].

### 4.2 Edge-based Distance Measure

LORS's quadratic complexity is not tractable for large-scale trajectory clustering. However, with certain relaxations, LORS is a viable solution, unlike other popularly used distance measures. Observe that in Figure 2,  $T_1$  and  $T_2$  there are two overlapping edges  $e_5$  and  $e_1$ , and the same score  $|e_5| + |e_1|$  can be computed using only an in-order intersection traversal *without* dynamic programming.

To verify this observation, we randomly conducted a million LORS computations using the **Porto** and **T-drive** datasets, and found **92.3%** and **91.3%** of trajectory computations using set intersection return identical scores to LORS, and the remaining **8%** have only a small average percent variance of **6.6%** and **2.7%** (see distribution in Figure 3), respectively. In order to achieve identical similarity scores, any two edges must have a stable successive co-occurrence relationship for any two paths in  $G$ , e.g.,  $e_5$  is before  $e_1$  in Figure 2, which co-occur in most trajectories. To formalize this observation, we introduce the following concept:

**DEFINITION 7. (Successive Probability  $\mathbb{P}(e_1, e_2)$ )** The successive probability of two edges  $e_1$  and  $e_2$  is computed as:

$$\mathbb{P}(e_1, e_2) = \frac{\max(C_o(e_1e_2), C_o(e_2e_1))}{C_o(e_1e_2) + C_o(e_2e_1)} \quad (3)$$

where  $C_o(e_1e_2)$  is the number of trajectories where  $e_1$  and  $e_2$  co-occur successively in  $D$ , i.e.,  $C_o(e_1e_2) = |\{D_s \subset D \mid \forall T \in D_s : T(e_1) < T(e_2)\}|$ ,  $T(e)$  is the order of  $e$  in  $T$ .

Based on these observations, we performed additional experiments to compute the successive probability distribution of the **Porto** dataset in order to further verify the viability of our new approach. There are 185,528,027 pairs of edges co-occurring in 1.56 million paths. Figure 3 shows that *around 65% of edge-pairs  $(e_1, e_2)$  have a stable successive relationship, i.e.,  $\mathbb{P}(e_1, e_2) = 1$ , and more than 80% have a probability  $\mathbb{P} > 0.8$* . After a visualized analysis of the paths over the map, we found the main reason for stable successive co-occurrence relationship is that most taxis follow the shortest

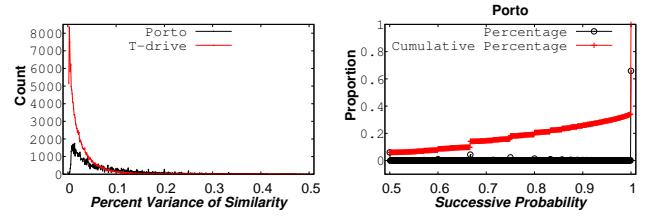


Figure 3: Distribution of variance & successive probability.

or fastest path between an origin and destination suggested by the navigation apps in practice (the percentage was reported as **96.8%** [63]). For successive probability less than 100%, it means that a path has a detour with a high chance. We thereby derive the following conjecture:

**CONJECTURE 1.** For any two trajectories  $T_1$  and  $T_2$  in the road network  $G$ , a  $\text{LORS}(T_1, T_2) = |T_1 \cap T_2|$  equivalent similarity score exists if  $T_1$  and  $T_2$  followed the shortest or fastest path to travel in  $G$ , where  $T_1 \cap T_2$  denotes the intersecting edges of  $T_1$  and  $T_2$ .

**PROOF.** The detailed proof is in our technical report [68], where we prove in two steps: (1) for any two edges  $e_1$  and  $e_2$ ,  $\mathbb{P}(e_1, e_2) = 1$  will always be true when all the trips are the shortest (fastest) paths in road network (weighted graph); (2)  $\text{LORS}(T_1, T_2) = |T_1 \cap T_2|$  holds when  $\mathbb{P}(e_1, e_2) = 1$ .  $\square$

Based on Conjecture 1 established under the above reasonable relaxations, we can define a new heuristic distance measure EBD using set intersection to support scalable  $k$ -paths clustering. Note that this is not the first time that a set intersection approach is used as a distance measure on sequential data – Peng et al. [62] quantized time series into a series of grid cells and performed set intersection to compute a distance. Specifically, we define EBD as follows:

$$\text{EBD}(T_1, T_2) = \max(|T_1|, |T_2|) - |T_1 \cap T_2| \quad (4)$$

where  $|T_1|$  is the travel length of the entire trajectory. To fit the  $\text{Dist}$  in Definition 6, we use trajectory travel length  $|T_1|$  and  $|T_2|$  to normalize the similarity to a distance value, similar to previous work [17, 79, 66]. Inspired by EDR [17] which bounds the distance value to  $[0, \max(|T_1|, |T_2|)]$ , we also choose the  $\max(|T_1|, |T_2|)$ , which can limit the length of centroid path  $\mu_j$  when minimizing the objective value in Equation 1. Moreover, EBD obeys the non-negativity, identity of indiscernibles, and symmetry [72].

**EXAMPLE 2.** As shown in Figure 2,  $T_1 = \{19, 3, 5, 1, 32, 6\}$  and  $T_2 = \{2, 5, 1, 4\}$ . We assume each edge has an equal length 1, then  $|T_1| = 6$ ,  $|T_2| = 4$  and they have two intersected edges  $\{e_1, e_5\}$ , the EBD distance between  $T_1$  and  $T_2$  is:  $\text{EBD}(T_1, T_2) = \max(6, 4) - 2 = 4$ .

### 4.3 Set Intersection with Sorted Lists

Computing the list intersection for two integer arrays has a complexity of  $\mathcal{O}(m^2)$  when these two sequences are unsorted, while two sorted lists will further reduce the complexity from quadratic to quasilinear ( $\mathcal{O}(m \log m)$ ).<sup>3</sup> This

<sup>3</sup>The exact cost is  $\mathcal{O}(m_1 \log(m_2/m_1))$  [20, 22, 67] using iterative binary search, where  $m_1$  and  $m_2$  are the length of the shorter and longer lists, as intersection requires  $m_1$  finger searches in the longer list. Linear expected time can be achieved using hash tables, but not with a comparison-based complexity model, and cannot be delta compressed. Next, we use  $\mathcal{O}(\text{EBD})$  as EBD's complexity.

is an order of magnitude improvement over unsorted lists, and the fast set intersection is a fundamental problem and being continuously explored [22, 67], which leaves space to improve the efficiency of EBD in the future.

Table 3: Time ( $\mu$ s) for pair-wise distance computations.

	EBD	LORS	EDR	DTW	Fré	Hau	ERP
Porto	0.88	19.5	20.2	107.3	180.9	125.1	172.5
T-drive	2.56	155.3	165.4	1940	3026	1991	2948

**Less Computation.** Table 3 compares six distance measures with EBD. We build the distance matrix of two datasets by setting  $|D| = 1000$  as other six measures are too slow to produce results when setting  $|D| = 10,000$ , and record the time on pair-wise distance computations. It shows that EBD is the fastest among all seven distance measures. EBD achieves two orders of magnitude improvement over the other six measures, especially for T-drive which stores long trajectories. For this collection, EBD only needs  $\frac{1}{80}$  of LORS and EDR’s time, and almost  $\frac{1}{1000}$  of the time on other measures.

**Less Space.** Sorting the trajectory lists not only accelerates the distance computation in Equation 4, but also reduces the storage costs as the data can be delta coded [82].

EXAMPLE 3. After sorting incrementally, we can compress the list using delta encoding [82] –  $T_1 = \{1, 3 - 1, 5 - 3, 6 - 5, 19 - 6, 32 - 19\} = \{1, 2, 2, 1, 13, 13\}$ , as shown in the bottom of Figure 2.

In our experiments on Porto, we show that the dataset can be compressed from 385MB to 178MB (see Table 5). Note that decompression is very efficient and will not affect the performance, as previously shown [69]. Hence, we preprocess all the trajectories by sorting in monotonically increasing order before clustering. It is not necessary to store the original trajectory, as the sorted array can be recovered easily using the graph connectivity information.

#### 4.4 Triangle Inequality

Fully metric distance measures are not generally required for trajectory-based pruning and indexing to be effective. However, obeying the triangle inequality can greatly improve commonly used pruning algorithms [28], and reduce the number of distance computations required in practice (see the detailed applications in Section 5.2). We prove that EBD guarantees to satisfy the triangle inequality.

LEMMA 1. For any trajectories  $T_1$ ,  $T_2$ , and  $T_3$ , we have:

$$\begin{aligned} EBD(T_1, T_2) + EBD(T_2, T_3) &\geq EBD(T_1, T_3) \\ |EBD(T_1, T_2) - EBD(T_2, T_3)| &\leq EBD(T_1, T_3) \end{aligned} \quad (5)$$

PROOF. The proof is in our technical report [68].  $\square$

Any metric index such as a VP-tree [77], M-tree [18], or MVP-tree [12] can be used to index the trajectories and support EBD for fast similarity search. In Section 6.2, we will propose a novel index-based batch pruning algorithm to further accelerate  $k$ -paths clustering.

### 5. BASELINE FOR K-PATHS

When using EBD, our baseline for  $k$ -paths works as shown in Algorithm 1. We first conduct the centroid initialization in line 1 (Section 5.1). Then in the first iteration ( $t = 0$ )

from line 4 to 9, we assign every trajectory  $T_i$  to the nearest centroid path. From the second iteration onward (line 12 to 25), we introduce two bounds between each trajectory and the centroid path to avoid unnecessary distance computations and accelerate assignments (Section 5.2). After the assignment in each iteration (line 27), we propose a solution to reduce the time complexity of refinement to linear with an objective function (Section 5.3).

#### 5.1 Centroid Initialization

Good initial clustering assignments can lead to faster convergence in  $k$ -means algorithms [10, 11]. We compare two different strategies: 1) randomly choosing  $k$  trajectories from dataset; 2) adopting  $k$ -means++ [10]. We find that random initialization is sufficient for fast convergence when using EBD for  $k$ -paths clustering (details in [68]). Hence, the centroid initialization of  $k$ -paths is not further explored in this work.

---

##### Algorithm 1: $k$ -paths ( $k$ , $D$ )

---

```

Input:  $k$ : #clusters,  $D$ : dataset.
Output:  $k$  centroid paths:  $\{\mu_1, \dots, \mu_k\}$ .
1  $t \leftarrow 0$ , initialize the centroid paths  $\mu = \{\mu_1, \dots, \mu_k\}$ ;
2 while  $\mu$  changed or  $t = 0$  do
3   if  $t = 0$  then
4     for every trajectory  $T_i \in D$  do
5        $min \leftarrow +\infty$ ;
6       for every centroid path  $\mu_j$  do
7          $lb(i, j) \leftarrow EBD(T_i, \mu_j)$ ;
8         if  $lb(i, j) < min$  then
9            $| a(i) \leftarrow j, min \leftarrow lb(i, j)$ ;
10          HISTOGRAMUPDATE( $a(i)$ , EH, ALH,  $T_i$ );
11    else
12      Update the centroid drift  $cd$  and bound  $cb$  for
      each cluster;
13      for every trajectory  $T_i \in D$  do
14        Update  $ub$  and  $lb$ ;
15        if  $\max(lb(i), \frac{cb(a'(i))}{2}) > ub(i)$  then
16           $| T_i$  stays in current cluster:  $a(i) \leftarrow a'(i)$ ;
17        else
18           $min \leftarrow +\infty$ ;
19          for every centroid path  $\mu_j$  do
20            if  $ub(i) > lb(i, j)$  then
21               $lb(i, j) \leftarrow EBD(T_i, \mu_j)$ ;
22              if  $lb(i, j) < min$  then
23                 $| a(i) \leftarrow j, min \leftarrow lb(i, j)$ ;
24            if  $a'(i) \neq a(i)$  then
25               $| HISTOGRAMUPDATE(a(i), EH, ALH, T_i)$ ;
26          for every centroid path  $\mu_j$  do
27            Compute  $O_j$  (Equation 10) and update  $\mu_j$ ;
28     $t \leftarrow t + 1$ ;
29 return  $\{\mu_1, \dots, \mu_k\}$ ;

```

---

#### 5.2 Trajectory Assignment

Based on the  $k$  newly chosen centroid paths, assigning each trajectory to the nearest cluster is called *trajectory assignment*. A baseline to achieve this is to compute its distance to every cluster’s centroid path  $\mu_j$ , as shown in line 4 to 9 of Algorithm 1 in the first iteration. The overall complexity of each iteration will be  $\mathcal{O}(nk) \times \mathcal{O}(EBD)$ .

**Pruning by Lower Bounds.** Reducing the complexity for a single distance computation can significantly increase the performance, while computing the distance with all centroid paths for every trajectory is still expensive. If we can reduce the number of distance computations, additional performance gains can be achieved. Computing lower bounds of distance without accessing the trajectory directly based on the triangle inequality [26] and the previous distance computation [23, 56] is a widely adopted method in  $k$ -means.

We now show how our bounding method to solve the EBD based  $k$ -paths. Let  $lb(i, j)$  denote the lower bound distance between a trajectory  $T_i$  (which was assigned to  $S_{a'(i)}$  in last iteration) and a centroid path  $\mu_j$  ( $1 \leq j \leq k$  and  $j \neq a'(i)$ ), and let  $ub(i)$  denote the upper bound distance between  $T_i$  and its nearest centroid path  $\mu_{a(i)}$ . An array is maintained to store the lower bound distance to all other clusters for each trajectory, each of which is initialized as the real distance in the first iteration of assignment (line 7).

**1) Centroid Drift.** For every trajectory  $T_i$ , we maintain (1) the lower bound distance to the previous centroid paths  $\mu'_j$  for all  $k - 1$  clusters, i.e.,  $lb(i, j) = EBD(T_i, \mu'_j)$ , where  $j \in [1, k]$  and  $j \neq a'(i)$ , and (2) an upper bound distance  $ub(i) = EBD(T_i, \mu'_{a'(i)})$ . After each refinement, the distance between the current centroid  $\mu_j$  and the previous centroid  $\mu'_j$  in cluster  $S_j$  will be computed as the *centroid drift*  $cd(j) = EBD(\mu_j, \mu'_j)$ . Before computing the distance between a trajectory and the new centroid path, we update the stored bounds with the centroid drift based on the triangle inequality, i.e.,  $lb(i, j) = |lb(i, j) - cd(j)|$ ,  $ub(i) = ub(i) + cd(a'(i))$ . As a result, it guarantees that  $T_i$  cannot be assigned to the centroid path  $\mu_j$  ( $a(i) \neq j$ ) if  $ub(i) < lb(i, j)$ , denoted as:

$$lb(i, j) > ub(i) : a(i) \neq j \quad (6)$$

In addition to maintaining a lower bound for every centroid path, the minimum lower bound is set as the *global lower bound*  $lb(i) = \min_{j \neq a'(i)} lb(i, j)$  distance for each trajectory  $T_i$ , i.e.,  $lb(i)$  is the lower bound distance from  $T_i$  to its second nearest cluster. Then,  $T_i$  can stay in cluster  $S_{a'(i)}$  (line 16) if  $lb(i) > ub(i)$ , denoted as:

$$lb(i) > ub(i) : a(i) = a'(i) \quad (7)$$

Note that  $lb(i, j)$  will be updated as  $EBD(T_i, \mu_j)$  if it is computed during the assignment (line 21); otherwise, we keep the current bound for next iteration. Same applies to  $ub(i)$  if  $EBD(T_i, \mu'_{a'(i)})$  is computed.

**2) Centroid Bound.** For every centroid path, we compute its distance to all other  $k - 1$  centroid paths and build the distance matrix over the centroid paths. It can be completed immediately as  $k$  is always small. Also, we store the minimum distance  $cb(a'(i)) = \min_{j \neq a'(i)} EBD(\mu'_{a'(i)}, \mu_j)$  as the global *filtering lower bound* (line 16), then we use the following comparisons to judge whether  $T_i$  should be assigned to  $S_j$  (line 20) or stay in cluster  $S_{a'(i)}$ :

$$\begin{aligned} \frac{EBD(\mu'_{a'(i)}, \mu_j)}{2} &> ub(i) : a(i) \neq j \\ \frac{cb(a'(i))}{2} &> ub(i) : a(i) = a'(i) \end{aligned} \quad (8)$$

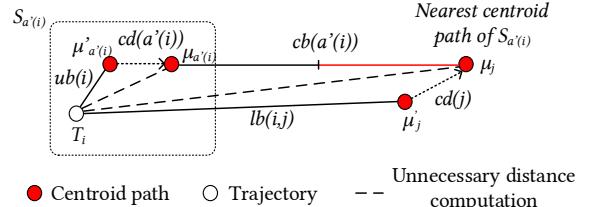


Figure 4: An example of lower bound and pruning.

Then, we combine two bounds to induce further pruning:

$$\max(lb(i), \frac{cb(a'(i))}{2}) > ub(i) : a(i) = a'(i) \quad (9)$$

EXAMPLE 4. In Figure 4,  $T_i$  was assigned to  $S_{a'(i)}$  in previous iteration.<sup>4</sup> Now the two new centroid paths are updated to new centroids, e.g.,  $\mu_j \rightarrow \mu_j'$ , and we need to assign  $T_i$  to a new centroid path. Instead of computing the distance  $EBD(T_i, \mu_{a'(i)})$  and  $EBD(T_i, \mu_j)$ , we compute the upper bound of  $EBD(T_i, \mu'_{a'(i)})$  as  $ub(i) = ub(i) + cd(a'(i))$ , and the lower bound of  $EBD(T_i, \mu_j)$  as  $lb(i, j) = |lb(i, j) - cd(j)|$  using the triangle inequality. If  $lb(i, j) > ub(i)$ , then  $EBD(T_i, \mu_j) > EBD(T_i, \mu'_{a'(i)})$ , which means  $T_i$  is closer to  $\mu'_{a'(i)}$  than  $\mu_j$ , and  $T_i$  can stay in cluster  $S_{a'(i)}$ . Another bound  $\frac{cb(a'(i))}{2}$  can be used to compare with  $ub(i)$  for pruning in a similar way.

### 5.3 Centroid Path Refinement

Similar to  $k$ -medoids [59], choosing the existing trajectory as the centroid path can make the result a real path in  $G$ . Such a trajectory  $T$  will minimize the distance to all other trajectories in the same cluster  $S_j$  ( $j \in [1, k]$ ), which can be denoted as:

$$O_j = \arg \min_{\mu_j \in S_j} \sum_{T \in S_j} EBD(T, \mu_j) \quad (10)$$

A naive way to update the centroid path is to pre-compute a distance matrix first, followed by an enumeration of each trajectory in the cluster. Then sum checking over all the trajectories in each cluster can be performed, and the trajectory with minimum distance as the new centroid path is chosen. The above baseline has a time complexity of  $\mathcal{O}(|S_j|^2) \times \mathcal{O}(EBD)$  where  $|S_j|$  is the number of trajectories in the cluster  $S_j$ . To reduce the complexity, we further transform the objective function to Equation 4:

$$\begin{aligned} O_j &= \arg \min_{\mu_j \in S_j} \left( \sum_{T \in S_j} \max(|T|, |\mu_j|) - \sum_{e \in \mu_j} \|e\| \right) \\ &= \arg \min_{\mu_j \in S_j} \left( \sum_{T \in S_j} |T| + \sum_{T \in S'_j} (|\mu_j| - |T|) - \sum_{e \in \mu_j} \|e\| \right) \\ &= \arg \min_{\mu_j \in S_j} (\|G_j\| + \sum_{T \in S'_j} (|\mu_j| - |T|) - \sum_{e \in \mu_j} \|e\|) \end{aligned} \quad (11)$$

where  $|T|$  is the length of trajectory  $T$ , and  $\|e\|$  is the frequency weight of the edge  $e$ , i.e., a product of the number of trajectories crossing  $e$  in cluster  $S_j$  and  $|e|$  (the length of  $e$ ).  $S'_j$  is a subset of  $S_j$  and stores all the trajectories with a length less than  $|\mu_j|$ .  $\|G_j\| = \sum_{T \in S_j} |T|$  is the weight of

<sup>4</sup>For a clearer observation of the pruning, the trajectory and EBD distance are simply drawn as point and line in a metric space.

frequency graph  $G_j$  for cluster  $S_j$  built from  $G$ , where the weight of each edge  $e \in G_j$  equals to  $EH_j(e)$  which is an *edge histogram* to be built.  $\|G_j\|$  is a constant and can be pre-computed by building a *length histogram*.

### 5.3.1 Histogram Construction

To compute the objective function for every trajectory in the cluster using Equation 11, we maintain two histograms for each cluster to update the centroid path in each iteration.

**Edge Histogram.** Given all trajectories in  $S_j$ , an edge histogram ( $EH_j$ ) for cluster  $S_j$  will store the frequency of edges in the graph, sorted in descending order.  $EH_j[l]$  returns the  $l$ -th largest frequency, and  $EH_j(e)$  returns the frequency of edge  $e$ , i.e.,  $EH_j(e) = \|e\|$ . We do not need to rebuild it in every iteration, but incrementally maintain one histogram for each cluster, and update it only when a trajectory moves into or out of this cluster (line 24). With more iterations, most trajectories will stay in the same cluster, so there will be fewer updates to the histogram. For all of the clusters, we also maintain a global edge histogram ( $EH_G$ ) for estimating the upper bound of the weight of each trajectory in Equation 14.

**Length Histogram.** The length histogram ( $LH$ ) mainly works for computing the second part of Equation 11. For each entry, the key is the length of trajectories (the standard unit is meter), the value is the number of trajectories having this length.  $LH$  is sorted by the key in ascending order,  $LH_j[l]$  returns the number of trajectories which have a length  $l$  in cluster  $S_j$ . With the built  $EH$  and  $LH$ , we further convert Equation 10 to:

$$\begin{aligned} O_j &= \arg \min_{\mu_j \in S_j} (\|G_j\| + \sum_{l=1}^{\lfloor \mu_j \rfloor} (|\mu_j| - l) LH_j[l] - \sum_{e \in \mu_j} EH_j(e)) \\ &= \arg \min_{\mu_j \in S_j} (\|G_j\| + ALH_j[\lfloor \mu_j \rfloor] - \sum_{e \in \mu_j} EH_j(e)) \end{aligned} \quad (12)$$

where  $ALH_j$  is the accumulated length histogram built from  $LH_j$  by:

$$ALH_j[m] = \begin{cases} 0, 1 \leq m \leq \min_{T \in S_j} |T| \\ ALH_j[m-1] + \sum_{1 \leq l \leq m} LH_j[l], \\ m \leq \max_{T \in S_j} |T| \end{cases} \quad (13)$$

### 5.3.2 Selecting Trajectories as Centroid Paths

By Equation 12, we use the edge and length histograms to check every trajectory in the cluster and find the one with the minimum objective value. This method reduces the complexity from  $\mathcal{O}(|S_j|^2) \times \mathcal{O}(\text{EBD})$  to  $\mathcal{O}(|S_j|)$  based on the incremental histograms maintained.

To this end, we still need to check the aggregated distance in Equation 12 of every trajectory to choose the minimum one, so the running time will increase w.r.t. the size of the data. Computing a lower bound for every trajectory  $T$  before computing the objective value by Equation 10 is a better approach. By using the sum of the  $|T|$  highest frequency of edges in the histogram of a cluster, the lower bound objective value for the trajectory can be computed, as shown below:

$$\sum_{e \in T} EH_j(e) < UB_1(T) = \sum_{e \in T} EH_G(e) \quad (14)$$

$$\sum_{e \in T} EH_j(e) < UB_2(T) = \sum_{1 \leq l \leq |T|} EH_j[l] \quad (15)$$

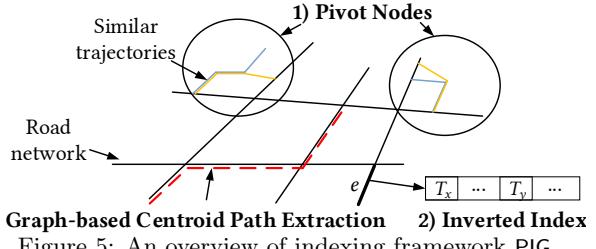


Figure 5: An overview of indexing framework PIG.

We can combine these two bounds as  $\max(UB_1(T), UB_2(T))$  to prune a trajectory before computing Equation 12.

## 5.4 Performance Discussions

The proposed baseline can avoid distance computations to some extent by using bounding and histogram techniques during assignments (Equation 6 and 9), and refinement (Equation 14 and 15). However, the algorithm still needs to scan every trajectory in the dataset, as shown in lines 4 and 26 of Algorithm 1. While scanning the entire dataset is not scalable in large collections, indexing techniques can be employed to avoid accessing the trajectory data.

Moreover, choosing an existing trajectory as a centroid path may reduce quality. For example, traffic cameras may not correctly read the plate number every time which leads to incomplete trajectories, or when an entire-day taxi trajectory is improperly segmented (which occurs in the T-drive dataset as shown in Figure 15(a) and (b)). To avoid such cases and make the refinement more robust, choosing a complete real path of moderate length is crucial.

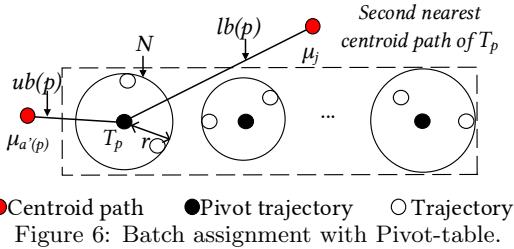
## 6 BOOSTING K-PATHS WITH PIG

In this section, we propose an indexing framework called PIG to further boost the performance. PIG is composed of three modules (Figure 5): a Pivot-table, an Inverted index, and a Graph traversal algorithm. In particular, the inverted index on each edge  $e$  of the road network can further reduce the distance computation cost; pivot nodes in the Pivot-table can bound a set of similar trajectories together instead of assigning them individually; the refinement step is converted to a more scalable graph traversal problem-CPEP to avoid repeated scanning of the trajectory dataset, where a robust, practical, and efficient greedy algorithm is proposed.

### 6.1 Inverted Index Acceleration

For all trajectories in the dataset, an inverted index is built where the key is the edge and the value is a sorted list of trajectory IDs passing this edge. The inverted index can avoid distance computations to accelerate the  $k$ -paths.

We first assign the trajectories that intersect with the centroid paths by computing the EBD distance and using the lower bound for pruning. For the remaining trajectories which do not intersect with any of the centroid paths, we assign each of them according to their lengths as the distance between  $T_i$  and the cluster will be  $\max(|T_i|, |\mu_j|)$ . If a trajectory  $T_i$  does not occur in any inverted list of a centroid path  $\mu_j$ , then we do not need to check the intersection  $|T_i \cap \mu_j|$  as it is equal to 0, and we can use  $\max(|T_i|, |\mu_j|)$  as the distance directly. Similarly, we can also build an inverted index on every vertex to accelerate the distance computation for EDR if it is used for road network trajectories. The inverted index can also allow us to interactively explore the trajectories in a specific range or path efficiently [70].



## 6.2 Pivot-table for Metric Feature

In this subsection, we will present an index which groups similar trajectories to accelerate the assignment step of the baseline proposed in Section 5. Before introducing our solution, it is noteworthy that Kanungo et al. [40] have earlier used a k-d tree and a perpendicular bisector to prune a group of points in nodes of the k-d tree in Euclidean space, and showed that it can greatly improve performance, especially in low dimensional space [34]. However, it cannot be used in a metric space covering EBD.

### 6.2.1 Pruning Mechanism

*Metric space indexing* is a widely explored area for fast similarity search. Among all metric indexing methods, a *pivot-based index* is one of the most popular choices [15]. The idea is to group a set of trajectories into several nodes. Inside each node  $N$ , a trajectory called *pivot*  $T_p$  is chosen, and every trajectory inside  $N$  has a distance less than a *radius*  $r$  to  $T_p$ . When a query trajectory  $q$  scans node  $N$ , the node will be pruned if  $\text{dist}(q, T_p) - r > \text{min}_{\text{dist}}(q)$ , where  $\text{min}_{\text{dist}}(q)$  is the current minimum distance. The assignment can be accelerated by pruning a group of trajectories.

**Grouping Trajectories.** Similarly, given a node  $N$  with a pivot trajectory  $T_p$  and radius  $r$ , we can have the following pruning rules with cluster  $S_j$  by extending Equation 6 & 8:

$$lb(p, j) - r > ub(p) + r : a(N) \neq j \quad (16)$$

$$\frac{\text{EBD}(\mu_{a'(p)}, \mu_j)}{2} - r > ub(p) + r : a(N) \neq j \quad (17)$$

Before comparing with every centroid path, we use a global bound similar to the one described in Equation 9.

$$\max(lb(p), \frac{cb(a'(p))}{2}) - r > ub(p) + r : a(N) = a'(p) \quad (18)$$

For the assignment process, we will assign a node of similar trajectories directly to cluster  $S_j$  if the gap between bounds is bigger than  $2r$ , as shown in Equation 18; otherwise, we scan the trajectories inside the node and proceed with assignments. The centroid paths  $\mu_j$  pruned by Equation 16 and 17 are no longer checked for the child trajectories of node  $N$ .

**EXAMPLE 5.** As shown in Figure 6, a pivot-table is created with a set of trajectory nodes represented as circles. Node  $N$  has a pivot trajectory  $T_p$ , and all trajectories inside the node have a distance less than a radius  $r$  to  $T_p$ .  $\mu_j$  is the second nearest centroid path to  $T_p$ . We can assign the whole node  $N$  to the cluster  $S_{a'(p)}$  if its pivot trajectory  $T_p$ 's upper bound plus the radius  $r$  (the greatest distance from any trajectory in  $N$  to  $\mu_{a'(p)}$ ) is smaller than the lower bound minus  $r$  (the smallest distance from any trajectory in  $N$  to  $\mu_j$ ), i.e.,  $lb(p) - r > ub(p) + r$ .

To group all trajectories into nodes, an M-tree [18] can be used. However, when millions of trajectories are inserted into the tree, it is inefficient, and nodes can have unacceptably large radii to cover the  $mc$  trajectories or sub-nodes, where  $mc$  is the minimum capacity of a node. This results in degraded pruning performance. From Equation 18, we can observe that a node with a large radius  $r$  rarely induces pruning, and we have to scan the child trajectories in that node if it is not pruned. Therefore, the tree structure is not used in this work. Instead, we use a lookup table containing all the *pivot nodes*.

### 6.2.2 Pivot-table Construction

As shown in Algorithm 2, we propose a novel index construction method for the  $k$ -paths clustering problem. First, all trajectories are divided into  $k$  clusters with  $k$ -paths (line 4), by employing the baseline in Section 5. For a cluster with more than  $k$  trajectories, we keep performing  $k$ -paths clustering until the number of trajectories inside is no larger than  $k$ . The final cluster will form a node and be added into the pivot table  $PT$ , and the centroid path  $\mu_j$  is the pivot trajectory for that node. After generating all of the nodes, the edge and length histograms are built for every node. This is used for refinement when a node is assigned to a cluster.

Algorithm 2 can also be used to discover the best  $k$  if we have a better idea on the capacity of each cluster than the parameter  $k$ , especially when clustering taxi trips to an unknown number of potential bus routes, where the capacity of a route is given. If a cluster has a number of trajectories greater than the capacity, we will divide this cluster into more sub-clusters, same as the pivot-table construction.

---

#### Algorithm 2: PIVOTTABLECONSTRUCTION( $k, D$ )

---

**Input:**  $k$ : #clusters,  $D$ : dataset.  
**Output:** pivot table  $PT$

```

1  $PT \leftarrow \emptyset, Q.push(D);$ 
2 while  $Q$  is not empty do
3    $D_s \leftarrow Q.poll();$ 
4    $S \leftarrow k\text{-paths}(k, D_s);$ 
5   for every cluster  $S_j \in S$  do
6     if  $|S_j| \leq k$  then
7        $r \leftarrow \text{GETRADIUS}(S_j, \mu_j);$ 
8        $PT.add(\mu_j, r, S_j);$ 
9     else
10       $Q.push(S_j);$ 
11 return  $PT;$ 

```

---

## 6.3 Graph-based Centroid Path Extraction

To further minimize the objective value in Equation 10, we define the following problem with an objective function to find a path in  $G$  instead of an existing trajectory.

**DEFINITION 8. (*Centroid Path Extraction Problem (CPEP)*)** Given a road network  $G$ , CPEP finds a path  $\mu_j^G$  in  $G$  to minimize the EBD with all the trajectories in the cluster  $S_j$ , which can be formulated as:

$$\begin{aligned} O_j^G &= \arg \min_{\mu_j^G \in G_j} \sum_{T \in S_j} \text{EBD}(T, \mu_j^G) \\ &= \arg \min_{\mu_j^G \in G_j} (\|G_j\| + \text{ALH}_j([\mu_j^G]) - \sum_{e \in \mu_j^G} \text{EH}_j(e)) \end{aligned} \quad (19)$$

where  $\mu_j^G \in G_j$  denotes that  $\mu_j^G$  is a path in the frequency graph  $G_j$  with a length  $|\mu_j^G| = [l_{min}, l_{max}]$ , and initially,  $l_{min} = \min_{T \in S_j} |T|$  and  $l_{max} = \max_{T \in S_j} |T|$ .

Minimizing this function can return a centroid path no worse than choosing an existing trajectory as the centroid path, because each trajectory in  $S_j$  is a path in  $G_j$ . Such a path is used to build the frequency graph  $G_j$  of  $S_j$ , so it can be found in  $G_j$  to achieve the same objective value in Equation 10. Moreover, a new path composed of the connected edges with high frequency in the graph can be scanned, so as to further reduce the objective value.

A straightforward method to answering CPEP is to find all the candidate paths in  $G$ . However, there are too many choices for a path with a length in the range  $[l_{min}, l_{max}]$ , and the brute force method cannot be resolved in the estimated time as CPEP is NP-hard by converting it to the  $k$  minimum Travel Salesman Problem ( $k$ -TSP) [29, 55].

**LEMMA 2.** *The CPEP which finds a path  $\mu_j^G$  in graph  $G_j$  for  $O_j^G$  is NP-hard.*

PROOF. The proof is in our technical report [68].  $\square$

The CPEP can be solved by setting different  $|\mu_j^G| \in [l_{min}, l_{max}]$  for  $k$ -TSP,<sup>5</sup> and then choosing the one with a minimum objective value. We can further narrow the range of  $|\mu_j^G|$  by decreasing  $l_{max}$  from  $\max_{T \in S_j} |T|$  to the first value that makes the objective value increase  $-ALH_j[l_{max}] - ALH_j[l_{max}-1] - EH_j[l_{max}] = \sum_{j=1}^{l_{max}} LH_j[j] - EH_j[l_{max}] > 0$ . When a  $\mu_j^G$  has a length of  $l_{max}$  and  $\sum_{j=1}^{l_{max}} LH_j[j] > EH_j[l_{max}]$ , the objective value will not decrease anymore. However, such a method still needs  $(l_{max} - l_{min} + 1)$  times of  $k$ -TSP. Hence, we develop an efficient growth-pruning search algorithm over the graph.

**A Greedy Algorithm for CPEP.** In Algorithm 3, we first initialize all edges of the candidate paths, then all of the paths are grown by appending neighbor edges using a breadth-first search. We call this process *growth*. *Pruning* is performed on a growing path if it cannot be the result based on the lower bound computed by appending highly weighted potential edges (we call this the *threshold potential*).

Note that for a candidate path, we assume that there are no cycles (cycle-free) as most real paths will not cover the same edge more than once, and it is also a requirement of the shortest path search problem [19]. We do not insert this kind of path into the priority queue. Moreover, a consistent path should be only expanded to a start and end edges, and the connectivity of these two edges (*be*, *ee*) can be retrieved from the graph  $G$ . Further, we check whether  $EH_j.contains(e)$ , i.e., whether there is a trajectory crossing  $e$ .

**1) Priority Queue.** A priority queue  $PQ$  is used to maintain the candidate paths  $ps$  with a threshold potential  $LB(ps)$ . Each path in the queue is polled by choosing the smallest threshold potential, and checked to see if the path should be further expanded with a new edge. If true, it is added to  $PQ$ . We insert the path candidates incrementally from each of the single edges initially, and poll the ones with the smallest threshold potential to check if it can beat the current best centroid path. To achieve a tighter threshold potential, the temporary best centroid path is initialized as  $\mu_j'$  from the previous iteration, as shown from line 1 to 6.

<sup>5</sup>Here,  $k$  is the number of edges, which is a different concept from the  $k$  in  $k$ -paths.

**2) Threshold Potential.** If the threshold potential of the extended path with a new edge is greater than the current best path's objective value  $min$ , we will not push this candidate path onto  $PQ$ . For every candidate path, we can compute the threshold potential of the new path by appending it to the best path (line 15), which is computed as:

$$LB(ps) = \|G_j\| - \sum_{e \in ps} EH_j(e) + \min_{l=|ps|+1}^{l_{max}} (ALH_j[l] - \phi(EH_j, l)) \quad (20)$$

where  $\phi(EH_j, l)$  is the maximum possible edge weight for the appending path with a length  $l_{max} - l$ , we can let  $\phi(EH_j, l) = \sum_{z=1}^{l_{max}-l} EH_j[z]$ .

---

**Algorithm 3: REFINECPEP( $G, \mu_j'$ )**

---

**Input:**  $G$ : graph,  $\mu_j'$ : previous centroid path.  
**Output:** the centroid path  $\mu_j$ .

```

1  $PQ \leftarrow \emptyset, B \leftarrow \emptyset, min \leftarrow O_j^G(\mu_j'), it \leftarrow 0, \mu_j \leftarrow \mu_j'$ ;
2 for each edge  $e \in EH_j$  do
3   if  $min > LB(e)$  then
4      $| PQ.push(e, LB(e));$ 
5 while  $PQ.\text{IsNotEmpty}()$  do
6    $(ps, LB(ps)) \leftarrow PQ.poll();$ 
7   if  $LB(ps) \geq min$  or  $it \leq it_{max}$  then
8     break;
9   for each neighbor edge  $e$  of  $ps$  do
10    if  $EH_j.contains(e)$  and  $e \notin ps$  then
11       $ps \leftarrow ps + e;$ 
12      Compute  $O_j^G(ps)$  by Equation 19;
13      if  $O_j^G(ps) < min$  then
14         $| min \leftarrow O_j^G(ps), \mu_j \leftarrow ps;$ 
15      Compute  $LB(ps)$  by Equation 20;
16      if  $min > LB(ps)$  then
17        if  $B(ps) > LB(ps)$  then
18           $| PQ.push(ps, LB(ps));$ 
19           $| B.add(ps.be, ps.ee, LB(ps));$ 
20       $it \leftarrow it + 1;$ 
21 return  $\mu_j$ ;

```

---

**3) Repeatability.** Moreover, to avoid repetitive scanning of paths which share the same start and end edges, a buffer  $B$  is created to store the signature of each checked candidate path, where the key is composed of the start and end edge IDs, and the value is the threshold potential. If any path being checked has the same start and end edge IDs, we compare the threshold potential  $LB(ps)$  with the bound in the buffer  $B(ps)$ . The checked path will be pruned if  $LB(ps) > B(ps)$ , as shown in line 17.

**4) Termination.** If the current minimum objective value  $min$  is greater than the next polled path's threshold potential (in Equation 20), the whole algorithm will terminate and return the best path (line 7), as all unscanned paths' best cases are impossible to beat the current best path. However, according to our experiments, the gap between the best and threshold potential can be hard to determine. Hence, we terminate if the best path does not change after a fixed number of iterations  $it_{max}$  (5000 is sufficient according to our experiments), and the experiment shows that we achieve a better objective value than Equation 10 within thousands of iterations, i.e., it can find a better path than the dataset scanning based refinement method.

## 7. EXPERIMENTS

**Goals.** We would like to conduct experiments to verify the superiority of EBD based  $k$ -paths over state-of-the-art [14] in efficiency, scalability of the PIG index proposed for  $k$ -paths, the effectiveness of EBD over state-of-the-art [14, 46], as well as the impact of choices of  $k$ .

**Datasets.** (1) **Porto** [6]: trips in the city of Porto for one year (from 01/07/2013 to 30/06/2014) performed by all the 442 taxis. 2) **T-drive** [80]: trips for 30,000 taxis in Beijing over three months. **Porto** and **T-drive** are mapped into road network using *GraphHopper* [4]. Table 4 shows the statistics of these two datasets, Figure 7 shows the underlying road networks [5], and Figure 8 shows the histograms of all the edge length of **Porto** and **T-drive**.

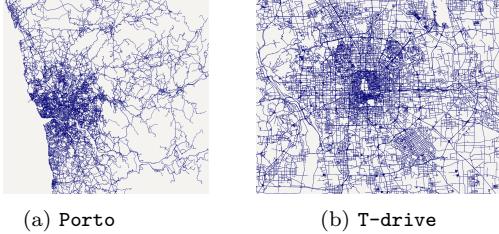


Figure 7: Road network overview of two collections.

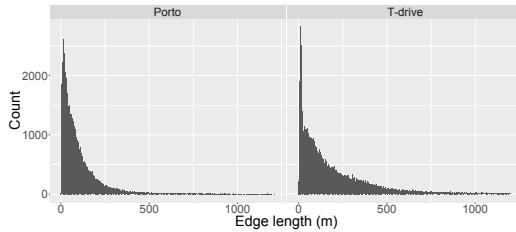


Figure 8: Edge length histograms of two datasets.

Table 4: Summary of datasets and road networks.

	Porto	T-drive
#trajectories	1,565,595	250,997
#total edges	100,995,114	59,360,981
#edges per trajectory	65	237
average travel length (m)	5632	31,056
Space (MB) of raw $D$	1853	752
#Edges	150,761	126,827
#Vertices	114,099	54,198
Average edge length (m)	119	217
Space (MB) of $G$	11	5

**Implementations.** All experiments were performed on a server using an Intel Xeon E5 CPU with 256 GB RAM running RHEL v6.3 Linux, implemented in Java 8.0. The JVM Heap size was set to 16GB. *HashMap* is used to store the trajectories and build the inverted index, and *Google Guava MultiSet* [2] is used to build the histograms.

**Comparisons & Measures.** Our primary baseline is the most cited trajectory clustering work **TRACLUS** [46] which is density-based, and the state-of-the-art  $k$ -center for point-based trajectories [14] which is partition-based. We also choose **LORS** and five existing trajectory distance measures widely used in partition-based clustering as shown in Table 1, and integrate them with  $k$ -paths. Specifically, DTW [75], Discrete Fréchet Distance (Fré for short) [7], and EDR [73], Hausdorff (Hau for short) [14], ERP [61] are compared with  $k$ -paths and EBD, where the starting vertices of edges embody the point-based trajectory. Running time,

case study, convergence (objective value and running time in each iteration), and pruning power (#pruned distance computations) are reported.

### 7.1 Efficiency Study of $k$ -paths

For fair comparisons, we run experiments on the same dataset with the same randomly selected initial centroid paths in each test for the efficiency validations. We randomly generate the seed pool which contains 200 groups of initial centroid paths, and each group includes 100 trajectories for each dataset. For all  $k < 100$ , we choose the first  $k$  trajectories from each group and run every comparison for each group, and finally record the average running time and the number of iterations (#iterations).

**$k$ -paths with Various Distance Measures.** We first compare the single-iteration running time of EBD-based  $k$ -paths with that of other distance measures based  $k$ -paths for all datasets. We set  $|D| = 1000$  due to poor efficiency of other measures. The EBD-based  $k$ -paths incorporates all of the optimizations proposed in this paper; ERP also uses our lower bounding approach described in Section 5.2 and LORS uses our inverted index approach from Section 6.1. We ignore EDR as it has similar running time with LORS due to the same computation manner using dynamic programming, while the only difference is that EDR is based on point while LORS is based on edges. All other distance measures employ basic assignments and refinements based on a distance matrix. We also show the number of iterations of all measures. The result is presented in Figure 9.

**$k$ -paths with EBD.** The histograms in Figure 9 show the time spent on assignment and refinement respectively, and compare four methods from left to right:

**1)**: PIG proposed in Section 6; **2)**: baseline solution proposed in Section 5; **3)**: Lloyd’s algorithm introduced in Section 3.3 (we ignore it when verifying  $k$  as it is not competitive); **4)**:  $k$ -center [14] which adopts the Hausdorff distance measure [57] and uses another partition-based clustering framework.  $k$ -center tries to bound all points in  $k$  circles, and minimize the sum of the radius. We test  $|D| = \{100, 1000, 10,000, \underline{100,000}, 1,000,000\}$  and  $k = \{10, 20, \underline{30}, 40, 50\}$ . The underlined number is the default value when testing multiple parameters.

**Breakdown of Assignment Process.** Figure 10 shows the number of distance calculations in the assignment step using our proposed solutions as they are gradually increased. We compare Lloyd’s algorithm (LL), Centroid drift (CD), Centroid bound (CB), Inverted index (II), and Pivot-table (PT). The number of relocated trajectories (line 24 of Algorithm 1) and histogram updating time in each iteration are also observed.

**Breakdown of Refinement Process.** Figure 11 compares the refinement time and objective value changes in each iteration. Here “Scanning” represents the baseline method proposed in Section 5.3. Refinement time changes with the iterations in all of the 200 groups of experiments, and the boxplot reinforces our belief that we can get stable performance using randomly chosen seeds. It shows that CPEP can find a better centroid path (a smaller objective value defined in Equation 19 than Equation 10’s), and the running time of refinement is also smaller than the full scan based method. Moreover, Figure 12 shows the refinement performance of the greedy algorithm for CPEP. We show the

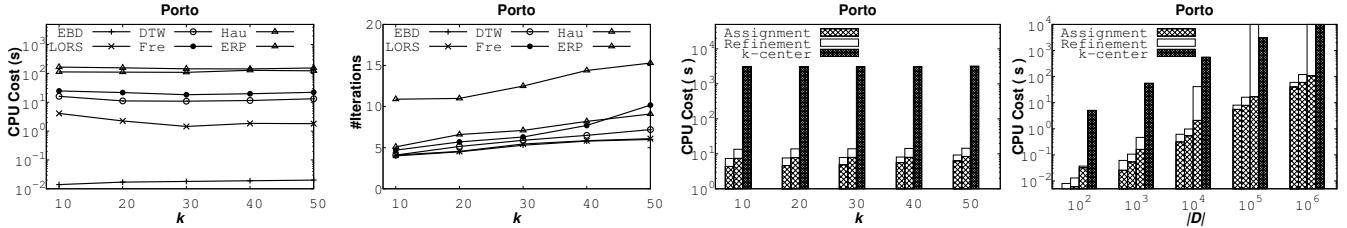


Figure 9: The single iteration clustering time and #iterations of 6 measures-based  $k$ -paths, EBD-based  $k$ -paths.

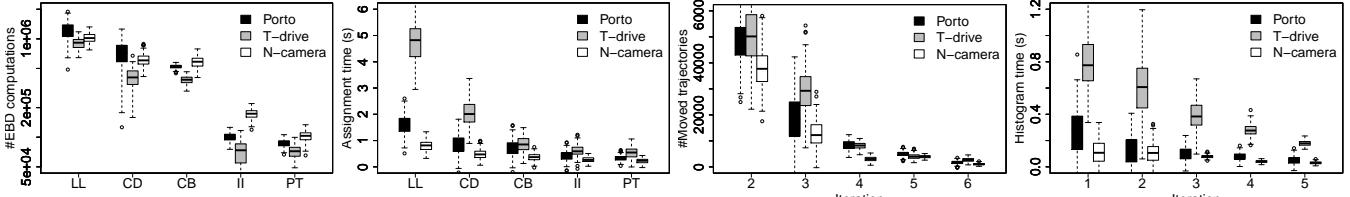


Figure 10: Assignment Breakdown: #EBD computation and time, #moved trajectories and histogram update time.

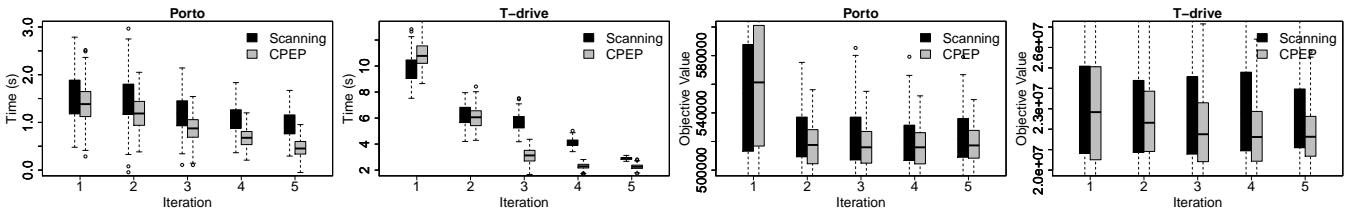


Figure 11: Refinement Breakdown: time and objective value in each iteration for scanning and CPEP.

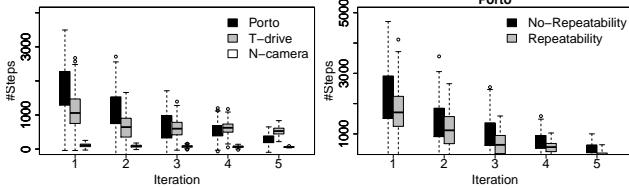


Figure 12: CPEP Breakdown: #steps in each iteration. average termination time (the steps used to update the  $\min$  in Algorithm 3) in each iteration of  $k$ -paths, which decreases with the iterations. This suggests that the optimal path can always be found in a limited number of path scans.

**OBSERVATION 1.** Firstly, *EBD* is the only distance measure which can achieve million-scale  $k$ -paths trajectory clustering for all measures, while the other measures can only achieve thousand-scale clustering.

Secondly,  $k$ -paths with *EBD* and our pruning ideas outperform the state-of-the-art method  $k$ -center [14] by around two orders of magnitude. Specifically,  $k$ -paths needs fewer distance computations (more than 80% are pruned) by using proposed bounds and index, the histograms and graph traversal not only accelerate the refinement, but also return better centroid paths with a smaller objective value.

Finally, as the dataset size grows, the running time increases linearly, and the number of iterations will slightly increase. An increasing  $k$  leads to a small rise in the number of iterations and running time.

## 7.2 Scalability Study of Index

**Index Construction.** Table 5 shows the time spent on index and histogram construction. To compress the dataset and index, we use an open source library *JavaFastPFOR* [3] to compress the sorted lists as shown in Example 3.

**Data Scale.** Since we do not have access to other larger real-world dataset, to test the scalability, we double the

Table 5: Building time (s), space & compression (MB).

	Porto			T-drive		
	Time	Space	Com	Time	Space	Com
Dataset	-	385	178	-	226	82
Inverted index	94	381	111	42	212	65
Pivot-table	3232	66	-	2346	11	-
EH	0.21	1.15	-	0.12	0.96	-
ALH	0.13	41	-	0.08	64	-

datasets and collect the running time, as shown in Figure 13.

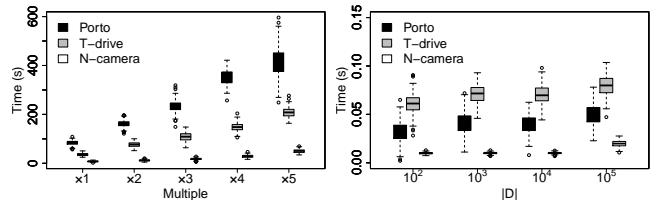


Figure 13: Scalability test on data scale and updates.

**$k$ -paths with Updates.** When we insert a new trajectory into the dataset, efficiently updating the centroid path is crucial. We can assign this new trajectory to  $k$  centroid paths using the lower bound technique, update the corresponding cluster's edge and length histograms, and then we can run CPEP and update the centroid path. We only insert new edges in the trajectory in the priority queue, as other edges have already been checked. Then the update cost is only related to  $k$ , and is independent of  $|D|$ .

**OBSERVATION 2.** The compression techniques help reduce the total trajectory dataset size from 385MB to 178MB for Porto, and from 226MB to 82MB for T-drive. Then, we get a compression ratio of 2.16 and 2.76, respectively. T-drive has a higher ratio as it has longer trajectories with 237 edges on average (see Table 4), where the compression will be more effective [82]. Hence, our algorithms are space-efficient and

scalable. Moreover, our algorithm efficiently supports updates in large collections. The running time grows linearly with the size of the data.

### 7.3 Effectiveness Study of $k$ -paths

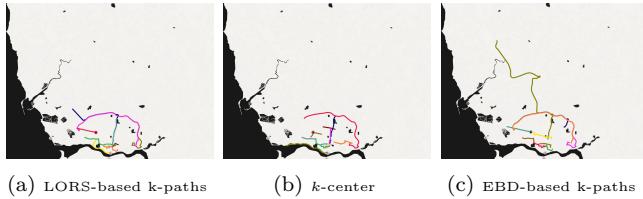


Figure 14: Case studies on Porto.

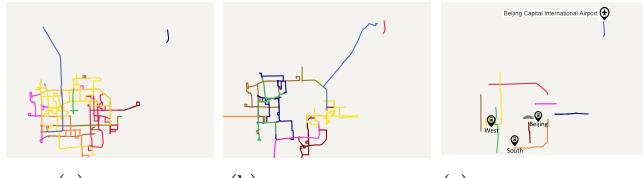


Figure 15: Case studies on T-drive.

#### 7.3.1 Case Study by Visualization

As shown in Figure 14, 15, and Figure 1 which we introduced in advance, we cluster the taxi datasets to help plan new bus routes. We compare five different methods: 1) TRACLUS; 2) LORS-based; 3) k-center [14]; 4) EBD-based without CPEP; 5) EBD-based with CPEP. Figure 1(a)(b) presents the density-based method TRACLUS [46]. The rest four are partition-based and presented in Figure 14, 15, and 1(c), and the output will be  $k$  paths with a different color. Note that we skip LORS-based for T-drive due to space limit.

Since the original TRACLUS algorithm cannot be used to cluster million-scale datasets, we optimize TRACLUS using our data modeling method. TRACLUS [46] is composed of two steps: partitioning and grouping. Partitioning finds the segments shared by trajectories, which can be seen as constructing the road network using trajectories. Since we use network-based trajectories, we can use the road network as the partition’s output directly. In grouping, we select all the edges with a frequency higher than a threshold (Lee et al. [46] set it as the average frequency by default), which can be done with our edge histogram.

In Figures 1(c) and 15(c) we highlight the locations of airport and main railway stations of each city. Beijing where the T-drive locates has three main railway stations, i.e., Beijing station, Beijing West, and Beijing South; Porto has two main railway stations, i.e., São Bento and Campanhã. We can observe that our cluster paths are often closely aligned with these locations (see the technical report for more detailed figures).

#### 7.3.2 Choice of $k$

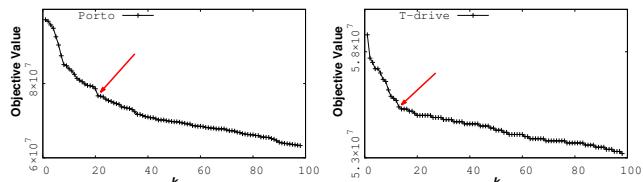


Figure 16: Finding the elbow of the objective value.

As  $k$  is the only parameter in the problem of  $k$ -paths, we also explore how to estimate the right  $k$  for each dataset. Using the elbow method [42, 44] to determine the optimal number of clusters is a common approach for  $k$ -means, i.e., finding the  $k$  where the gradient of objective value (Equation 1) starts to decrease. As shown in Figure 16, we increase  $k$  to observe the objective value changes in two datasets.

**OBSERVATION 3.** Firstly, Figure 1, 14, and 15 both show that the returned centroid paths with the EBD based  $k$ -paths covering the path beginning from or ending at the airport while the other two methods (LORS and  $k$ -center) do not. LORS performs not well as we can only use it to cluster 1000 trajectories. This is consistent with our common sense that taxis are one of the primary modes of transportation around the airports and railway stations, which also validates that our clustering result can show real demands, and help guide a transportation department in designing routes.

Secondly, by comparing Figure 1(c) which is CPEP-based  $k$ -paths and Figure 14(c), we can see that they are very similar visually, i.e., CPEP can find centroid paths as effective as scanning every trajectory. In Figure 15(b)(c), since T-drive stores the complete trajectory of every vehicle for long durations (while every trajectory in Porto is a single trip), choosing the existing trajectories as the centroid paths in  $k$ -paths produces long and chaotic trajectories, which are not as easy to work with as Porto is for visualizations. In contrast, CPEP based  $k$ -paths chooses the paths from the graph, which is much clearer and validates the robustness of our algorithm for CPEP which we believed is crucial (see Section 5.4).

Finally, Figure 16 shows that the selection of  $k$  based on finding the elbow is feasible for our two datasets. For Porto, the elbow is around  $k = 21$  (where the arrow points to), and the elbow is quite clear for T-drive, at around  $k = 13$ .

**Remarks.** More effectiveness studies can be explored using our demo system with interactive visualizations<sup>6</sup> [70] (see Scenario 2 of [70] for instructions, the response time may vary due to network delay). We also employ the *cluster validity index* (CVI) [49] for vector data to compare the cluster quality with existing distance measures based  $k$ -paths. It shows EBD performs well for most quality measures, the details can be found in our technical report [68].

## 8. CONCLUSIONS

In this paper, we proposed and studied  $k$ -paths— a fundamental trajectory clustering problem. To answer  $k$ -paths efficiently, we model trajectory data based on the road network and proposed a distance measure called EBD, which reduces the time complexity of distance computation and obeys the triangle inequality. Based on EBD, effective lower bound prunings and built histograms,  $k$ -paths can be answered using the classic Lloyd’s clustering algorithm efficiently. To further resolve the problem of 1:1 data access during assignment and refinement, we proposed an indexing framework called PIG that groups trajectories in a Pivot-table and builds inverted index avoids unnecessary distance computation, traverses a graph to find robust centroid paths. In the future, we will further explore the distributed  $k$ -paths problem to further scale out our approach – perhaps even reaching the billion-scale trajectory clustering milestone.

<sup>6</sup><http://115.146.93.77:8080/TorchServer/>

## 9. REFERENCES

- [1] Connected cars will send 25 gigabytes of data to the cloud every hour. <https://perma.cc/BPM2-QNW4>.
- [2] Guava: Google Core Libraries for Java. <https://github.com/google/guava>.
- [3] JavaFastPFOR: A simple integer compression library in Java. <https://github.com/lemire/JavaFastPFOR>.
- [4] Map Matching based on GraphHopper. <https://github.com/graphhopper/map-matching>.
- [5] OpenStreetMap. <https://www.openstreetmap.org>.
- [6] Taxi Service Trajectory (TST) Prediction Challenge 2015. <http://www.geolink.pt/ecmlpkdd2015-challenge/>.
- [7] P. K. Agarwal, K. Fox, K. Munagala, A. Nath, J. Pan, and E. Taylor. Subtrajectory clustering: Models and algorithms. In *PODS*, pages 75–87, 2018.
- [8] G. Andrienko, N. Andrienko, S. Rinzivillo, M. Nanni, D. Pedreschi, and F. Giannotti. Interactive visual clustering of large collections of trajectories. In *VAST*, pages 273–274, 2009.
- [9] S. Arora and G. Karakostas. A  $2 + \epsilon$  approximation algorithm for the k-MST problem. *Math. Program.*, 107(3):491–504, 2006.
- [10] D. Arthur and S. Vassilvitskii. K-Means++: The advantages of careful seeding. In *SODA*, pages 1027–1025, 2007.
- [11] B. Bahmani, R. Kumar, and S. Vassilvitskii. Scalable K-Means++. *PVLDB*, 5(7):622–633, 2012.
- [12] T. Bozkaya and M. Ozsoyoglu. Indexing large metric spaces for similarity search queries. *ACM Transactions on Database Systems*, 24(3):361–404, 1999.
- [13] T. Calinski and J. Harabasz. A dendrite method for cluster analysis. *Communications in Statistics*, 3(1):1–27, 1974.
- [14] T.-h. H. Chan, A. Guerquin, and M. Sozio. Fully dynamic k-center clustering. In *WWW*, pages 579–587, 2018.
- [15] L. Chen, Y. Gao, B. Zheng, C. S. Jensen, H. Yang, and K. Yang. Pivot-based metric indexing. *PVLDB*, 10(10):1058–1069, 2017.
- [16] L. Chen and R. Ng. On the marriage of Lp-norms and edit distance. In *VLDB*, pages 792–803, 2004.
- [17] L. Chen, M. T. Özsu, and V. Oria. Robust and fast similarity search for moving object trajectories. In *SIGMOD*, pages 491–502, 2005.
- [18] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *VLDB*, pages 426–435, 1997.
- [19] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to algorithms*. MIT press, 2009.
- [20] J. S. Culpepper and A. Moffat. Efficient set intersection for inverted indexing. *ACM Transactions on Information Systems*, 29(1):1–25, 2010.
- [21] D. L. Davies and D. W. Bouldin. A cluster separation measure. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1(2):224–227, 1979.
- [22] B. Ding and A. C. König. Fast set intersection in memory. *PVLDB*, 4(4):255–266, 2011.
- [23] Y. Ding, Y. Zhao, X. Shen, M. Musuvathi, and T. Mytkowicz. Yinyang K-means: A drop-in replacement of the classic K-means with consistent speedup. In *ICML*, pages 579–587, 2015.
- [24] J. C. Dunn. Well-separated clusters and optimal fuzzy partitions. *Journal of Cybernetics*, 4(1):95–104, 1974.
- [25] T. Eiter and H. Mannila. Computing discrete Fréchet distance. Technical report, 1994.
- [26] C. Elkan. Using the triangle inequality to accelerate k-means. In *ICML*, pages 147–153, 2003.
- [27] N. Ferreira, J. T. Kłosowski, C. E. Scheidegger, and C. T. Silva. Vector field k-means: Clustering trajectories by fitting multiple vector fields. *Computer Graphics Forum*, 32(3):201–210, 2013.
- [28] V. Ganti, R. Ramakrishnan, J. Gehrke, A. Powell, and J. French. Clustering large datasets in arbitrary metric spaces. In *ICDE*, pages 502–511, 1999.
- [29] N. Garg. Saving an Epsilon: a 2-approximation for the k-MST problem in graphs. In *STOC*, pages 396–402, 2005.
- [30] J. Gudmundsson and N. Valladares. A GPU approach to subtrajectory clustering using the Fréchet distance. In *SIGSPATIAL*, pages 259–268, 2012.
- [31] Y. D. Gunasekaran, M. F. Rahman, S. Hasani, N. Zhang, and G. Das. DBLOC: Density based clustering over location based services. In *SIGMOD*, pages 1697–1700, 2018.
- [32] D. Habich, M. Hahmann, and W. Lehner. Parameterfreies Clustering mittels Multi-Level Ansatz. In *LWA*, pages 36–43, 2007.
- [33] M. Halkidi, Y. Batistakis, and M. Vazirgiannis. On clustering validation techniques. *Journal of Intelligent Information Systems*, 17(2–3):107–145, 2001.
- [34] G. Hamerly. Making k-means even faster. In *SDM*, pages 130–140, 2010.
- [35] B. Han, L. Liu, and E. Omiecinski. Road-network aware trajectory clustering: Integrating locality, flow, and density. *IEEE Transactions on Mobile Computing*, 14(2):416–429, 2015.
- [36] Z. Hong, Y. Chen, and H. S. Mahmassani. Recognizing network trip patterns using a Spatio-Temporal vehicle trajectory clustering algorithm. *IEEE Transactions on Intelligent Transportation Systems*, 19(8):2548–2557, 2018.
- [37] C. C. Hung, W. C. Peng, and W. C. Lee. Clustering and aggregating clues of trajectories for mining trajectory patterns and routes. *VLDB Journal*, 24(2):169–192, 2015.
- [38] Z. G. Ives. Technical perspective - k-Shape : Efficient and accurate clustering of time series. *SIGMOD Record*, 45(1):68, 2016.
- [39] C. S. Jensen, D. Lin, and B. C. Ooi. Continuous clustering of moving objects. *IEEE Transactions on Knowledge and Data Engineering*, 19(9):1161–1173, 2007.
- [40] T. Kanungo, S. Member, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, A. Y. Wu, and S. Member. An efficient k-means clustering algorithm: Analysis and implementation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(7):881–892, 2002.
- [41] E. Keogh. Exact indexing of dynamic time warping. In *VLDB*, pages 406–417, 2002.
- [42] D. J. J. Ketchen and C. L. Shook. The application of cluster analysis in strategic management research: an analysis and critique. *Strategic Management Journal*, 17(6):441–458, 1996.
- [43] A. Kharrat, I. Popa, K. Zeitouni, and S. Faiz. Clustering algorithm for network constraint trajectories. In *SDH*, pages 631–647, 2008.
- [44] T. M. Kodinariya and P. R. Makwana. Review on determining number of cluster in K-Means clustering. *International Journal of Advance Research in Computer Science and Management Studies*, 1(6):2321–7782, 2013.
- [45] S. Koide, Y. Tadokoro, C. Xiao, and Y. Ishikawa. CiNCT: Compression and retrieval for massive vehicular trajectories via relative movement labeling. In *ICDE*, pages 1097–1108, 2018.
- [46] J.-g. Lee, J. Han, and K.-Y. Whang. Trajectory clustering: A partition-and-group framework. In *SIGMOD*, pages 593–604, 2007.
- [47] Z. Li, J.-G. Lee, X. Li, and J. Han. Incremental clustering for trajectories. In *DASFAA*, pages 32–46, 2010.
- [48] D. Liu, D. Weng, Y. Li, J. Bao, Y. Zheng, H. Qu, and Y. Wu. SmartAdP: Visual analytics of large-scale taxi trajectories for selecting billboard locations. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):1–10, 2017.
- [49] Y. Liu, Z. Li, H. Xiong, X. Gao, J. Wu, and S. Wu. Understanding of internal clustering validation measures. In *ICDM*, pages 911–916, 2010.

- [50] S. P. Lloyd. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28(2):129–137, 1982.
- [51] Y. Lou, C. Zhang, Y. Zheng, X. Xie, W. Wang, and Y. Huang. Map-matching for low-sampling-rate GPS trajectories. In *SIGSPATIAL*, pages 352–361, 2009.
- [52] W. Luo, H. Tan, L. Chen, and L. M. Ni. Finding time period-based most frequent path in big trajectory data. In *SIGMOD*, pages 713–724, 2013.
- [53] J. Macqueen. Some methods for classification and analysis of multivariate observations. *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, 1(233):281–297, 1967.
- [54] B. Morris and M. Trivedi. Learning trajectory patterns by clustering: Experimental studies and comparative evaluation. In *CVPR*, pages 312–319, 2009.
- [55] V. Nagarajan and R. Ravi. The directed orienteering problem. *Algorithmica*, 60(4):1017–1030, 2011.
- [56] J. Newling and F. Fleuret. Fast k-means with accurate bounds. In *ICML*, pages 936–944, 2016.
- [57] S. Nutanong, E. H. Jacox, and H. Samet. An incremental Hausdorff distance calculation algorithm. *PVLDB*, 4(8):506–517, 2011.
- [58] J. Paparrizos and L. Gravano. k-Shape: Efficient and accurate clustering of time series. In *SIGMOD*, pages 1855–1870, 2015.
- [59] H. S. Park and C. H. Jun. A simple and fast algorithm for K-medoids clustering. *Expert Systems with Applications*, 36(2):3336–3341, 2009.
- [60] J. G. Pearce, Z. Shaar, and R. E. Crosbie. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, 20:53–65, 1987.
- [61] N. Pelekis, I. Kopanakis, E. E. Kotsifakos, E. Frentzos, and Y. Theodoridis. Clustering trajectories of moving objects in an uncertain world. In *ICDM*, pages 417–427, 2009.
- [62] J. Peng, H. Wang, J. Li, and H. Gao. Set-based similarity search for time series. In *SIGMOD*, pages 2039–2052, 2016.
- [63] C. G. Prato and S. Bekhor. Applying Branch-and-Bound Technique to Route Choice Set Generation. *Transportation Research Record: Journal of the Transportation Research Board*, 1985(1):19–28, jan 2006.
- [64] E. Rendón, I. Abundez, A. Arizmendi, and E. M. Quiroz. Internal versus External cluster validation indexes. *International Journal of Computers and Communications*, 5(1):27—34, 2011.
- [65] R. Song, W. Sun, B. Zheng, and Y. Zheng. PRESS: A novel framework of trajectory compression in road networks. *PVLDB*, 7(9):661–672, 2014.
- [66] M. Vlachos, G. Kollios, and D. Gunopulos. Discovering similar multidimensional trajectories. In *ICDE*, pages 673–684, 2002.
- [67] Q. Wang and T. Suel. Document Reordering for Faster Intersection. *PVLDB*, 12(5):475–487, 2019.
- [68] S. Wang, Z. Bao, J. S. Culpepper, T. Sellis, and X. Qin. Fast Large-Scale Trajectory Clustering (Technical Report). <https://github.com/tgbnhy/k-paths-clustering/blob/master/k-paths-vldb.pdf>.
- [69] S. Wang, Z. Bao, J. S. Culpepper, Z. Xie, Q. Liu, and X. Qin. Torch: A search engine for trajectory data. In *SIGIR*, pages 535–544, 2018.
- [70] S. Wang, Y. Shen, Z. Bao, and X. Qin. Intelligent traffic analytics: from monitoring to controlling. In *WSDM*, pages 778–781, 2019.
- [71] D. Wen, L. Qin, Y. Zhang, L. Chang, and X. Lin. Efficient structural graph clustering: An index-based approach. *PVLDB*, 11(3):243–255, 2017.
- [72] Wikipedia contributors. Metric space — Wikipedia, the free encyclopedia. [https://en.wikipedia.org/w/index.php?title=Metric\\_space&oldid=889625849](https://en.wikipedia.org/w/index.php?title=Metric_space&oldid=889625849), 2019. [Online; accessed 30-April-2019].
- [73] Y. Wu, H. Shen, and Q. Z. Sheng. A cloud-friendly RFID trajectory clustering algorithm in uncertain environments. *IEEE Transactions on Parallel and Distributed Systems*, 26(8):2075–2088, 2015.
- [74] X. Yang, B. Wang, K. Yang, C. Liu, and B. Zheng. A Novel Representation and Compression for Queries on Trajectories in Road Networks. *IEEE Transactions on Knowledge and Data Engineering*, 30(4):613–629, 2018.
- [75] D. Yao, C. Zhang, Z. Zhu, J. Huang, and J. Bi. Trajectory clustering via deep representation learning. In *IJCNN*, pages 3880–3887, 2017.
- [76] B.-k. Yi and F. Park. Efficient Retrieval of Similar Time Sequences under Time Warping. In *ICDE*, pages 201–208, 1997.
- [77] P. Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. In *SODA*, pages 311–321, 1993.
- [78] G. Yuan, P. Sun, J. Zhao, D. Li, and C. Wang. A review of moving object trajectory clustering algorithms. *Artificial Intelligence Review*, 47(1):123–144, 2017.
- [79] H. Yuan and G. Li. Distributed In-Memory Trajectory Similarity Search and Join on Road Network. In *ICDE*, pages 1262–1273, 2019.
- [80] J. Yuan, Y. Zheng, C. Zhang, W. Xie, X. Xie, G. Sun, and Y. Huang. T-drive: Driving directions based on taxi trajectories. In *GIS*, pages 99–108, 2010.
- [81] P. Zhang, Z. Bao, Y. Li, G. Li, Y. Zhang, and Z. Peng. Trajectory-driven influential billboard placement. In *KDD*, pages 2748–2757, 2018.
- [82] J. Zobel and A. Moffat. Inverted files for text search engines. *ACM Computing Surveys*, 38(2):1–56, 2006.

# Appendices

## A Computation of LORS

Table 6 shows the similarity computation of LORS between  $T_1$  and  $T_2$  based on dynamic programming. For convenience of description, we assume that each edge  $e$  has a length  $|e| = 1$ , and apply the award policy by adding 1 to the similarity when two edges are the same.

Table 6: LORS computation with dynamic programming.

		19	3	5	1	32	6
	0	0	0	0	0	0	0
2	0	0	0	0	0	0	1
5	0	0	0	1	1	1	1
1	0	0	0	1	2	2	2
4	0	0	0	1	2	2	2

## B Proof of Conjecture 1

PROOF. We assume that a road network  $G$  is a symmetric directed graph with weighted edges, i.e., any two connected vertices have two edges with the same weight (bi-directed). For such a road network, the shortest path distance from origin vertex  $v_o$  to destination vertex  $v_d$  is the same with the distance from  $v_d$  to  $v_o$ . The weight can be either the travel length of edges for the shortest path, or the travel time (every edge has a travel time according to its speed limit) for the fastest path.

First, successive relationships between two connected edges are determined. Assume there are two edges  $AB$  and  $CD$  which are not directly connected, and they locate in a path  $T_1 = \{v_{o1}, A, B, C, D, v_{d1}\}$ . Then in  $T_1$ ,  $AB$  is in front of  $CD$ , the travel length from  $A$  to  $D$  is  $|AB| + |\overline{BC}| + |CD|$ , as shown in Figure 17, where the dotted line with three plus signs represents the detailed path between two vertices, such as  $B$  and  $C$ , we use  $\overline{BC}$  to denote the potential path. We prove it by contradiction, i.e., there exists another path  $T_2 = \{v_{o2}, C, D, A, B, v_{d2}\}$  where  $CD$  is in front of  $AB$ , the travel length from  $C$  to  $B$  is  $|CD| + |\overline{DA}| + AB$ . Further, we have  $|AB| + |\overline{BC}| + |CD| \leq |AD| = |\overline{DA}|$ , because in the shortest path search from  $v_{o1}$  to  $v_{d1}$ ,  $\{AB, BC, CD\}$  has been chosen as the sub-path between  $A$  and  $D$  with the minimum distance to travel. Similarly, we have  $|CD| + |\overline{DA}| + |AB| \leq |\overline{BC}|$ . We sum up these two inequalities and get  $2 * (|AB| + |CD|) \leq 0$  which violates the fact that edge's length is bigger than 0, then the counter example is not established, and edge  $AB$  will always locate before edge  $CD$  in any shortest path, i.e.,  $\mathbb{P}(AB, CD) = 1$ . Since  $AB$  and  $CD$  are two arbitrary edges in  $G$ , then we have  $\mathbb{P}(e_1, e_2) = 1$ .

With  $\mathbb{P}(e_1, e_2) = 1$ , for any two paths  $T_1$  and  $T_2$ , all their common edges will have the same successive order. For the edges that are not in the common edge set  $T_1 \cap T_2$ , since they will not affect the LORS similarity score (the similarity score will increase only when two matching edges are the same, see Table 6), we will get two sub-trajectories which are exactly the same after removing these edges from  $T_1$  and  $T_2$ . Then, we can return the length of this sub-trajectory as the similarity, i.e., the sum of its length  $|T_1 \cap T_2|$  without using dynamic programming in Table 6. Hence,  $\text{LORS}(T_1, T_2) = |T_1 \cap T_2|$  is proved.  $\square$

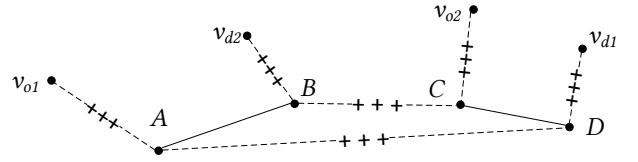


Figure 17: An example of edges' successive stability.

## C Proof of Lemma 1

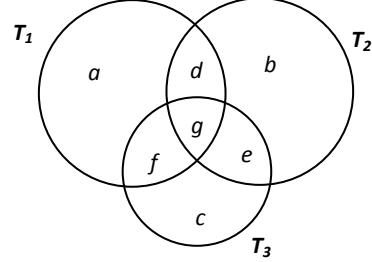


Figure 18: An example for metric proof, where each circle denotes the length, such as  $|T_1| = a + d + g + f$ .

PROOF. Figure 18 shows the intersections  $a, b, c, d, e, f, g$  among three trajectories, all of which have a value no less than 0.  $\text{EBD}(T_1, T_2) = \max(a + f, b + e)$ ,  $\text{EBD}(T_2, T_3) = \max(b + d, c + f)$ ,  $\text{EBD}(T_1, T_3) = \max(a + d, c + e)$ . We will prove  $\text{EBD}(T_1, T_2) + \text{EBD}(T_2, T_3) \geq \text{EBD}(T_1, T_3)$ ,  $\max(a + f, b + e) + \max(b + d, c + f) \geq \max(a + d, c + e)$ , and  $|\text{EBD}(T_1, T_2) - \text{EBD}(T_2, T_3)| \leq \text{EBD}(T_1, T_3)$ ,  $|\max(a + f, b + e) - \max(b + d, c + f)| \leq \max(a + d, c + e)$ . Since  $\max(a + f, b + e) + \max(b + d, c + f) \geq \max(a + f + b + d, b + e + c + f) \geq \max(a + d, c + e)$ , then  $\text{EBD}(T_1, T_2) + \text{EBD}(T_2, T_3) \geq \text{EBD}(T_1, T_3)$  is proved.

For  $|\max(a + f, b + e) - \max(b + d, c + f)|$ , we can assume  $a + f \geq b + e$  and  $b + d \geq c + f$  as there is no limitation on the value, then by combining two inequalities we have  $a + d \geq c + e$ . Now we have  $|\max(a + f, b + e) - \max(b + d, c + f)| = |a + f - b - d|$ ,  $\max(a + d, c + e) = a + d$ , and we need to prove  $|a + f - b - d| \leq a + d$ . Now we can explore two cases: 1) when  $a + f \geq b + d$ , then  $|a + f - b - d| - a - d = a + f - b - d - a - d = f - b - 2d$ , since we have assumed  $b + d \geq c + f$ , then  $b + d \geq f - d$ ,  $f - d - (b + d) = f - b - 2d \geq 0$ , and  $|a + f - b - d| - a - d \geq 0$ ; 2) when  $a + f \leq b + d$ ,  $|a + f - b - d| - a - d = b + d - a - f - a - d = b - 2a - f$ , since we have assumed  $b + e \leq a + f$  initially, then  $b - a \leq f - e$ , and  $b - 2a - f \leq f - e - a - f = -a - e \leq 0$ . So,  $|\text{EBD}(T_1, T_2) - \text{EBD}(T_2, T_3)| \leq \text{EBD}(T_1, T_3)$  is proved.  $\square$

## D Proof of Lemma 2

PROOF. We can convert Equation 19 as follows:

$$\begin{aligned} O_j^G &= \arg \min_{\mu_j^G \in G_j} (\|G_j\| + ALH_j[\mu_j^G] - \max \sum_{e \in \mu_j^G} EH_j(e)) \\ &= \arg \min_{|\mu_j^G|} (\|G_j\| + ALH_j[\mu_j^G] - kTSP(|\mu_j^G|, G_j)) \end{aligned} \quad (21)$$

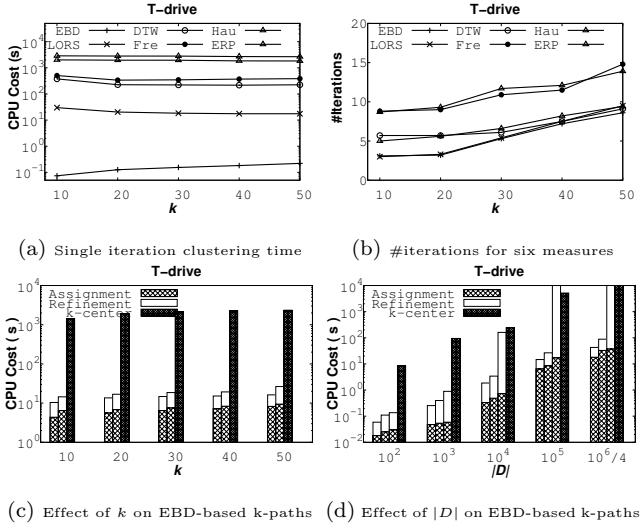


Figure 19: Efficiency studies on T-drive.

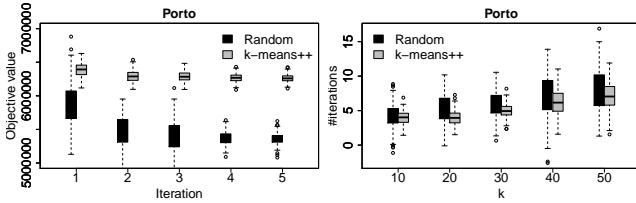


Figure 20: The effect of centroid path initialization.

where  $kTSP(|\mu_j^G|, G_j) = \max \sum_{e \in \mu_j^G} EH_j(e)$  outputs the maximum sum of the frequency of a path with a given length of  $|\mu_j^G|$  in graph  $G_j$ , which finds the path of length  $|\mu_j^G|$  in graph  $G_j$  with the greatest weight, which is an NP-hard problem:  $k$  minimum Travel Salesman Problem ( $k$ -TSP) [29, 9, 55] (Given a weighted graph, find a path of minimum<sup>7</sup> weight that passes through any  $k$  vertices, where we can relax our problem by setting the length of each edge as 1 similar to Table 6). Hence, CPEP is NP-hard.  $\square$

Note that a bounded approximation ratio for  $k$ -TSP can be obtained only under the assumption that the edge-lengths satisfy the triangle inequality [29, 9], since the length in our graph  $G_j$  is the frequency of edges crossing trajectories, and does not satisfy the triangle inequality, then no known bounded approximate solutions currently exist for CPEP.

## E More Experiments on T-drive and Centroid Initialization

Figure 19 shows an additional efficiency study for  $k$ -paths on T-drive. We can observe similar trends to the Porto dataset in Figure 9.

Figure 20 shows comparisons with  $k$ -means++ [10] in terms of the number of iterations and increasing  $k$ , and the objective values as the iterations increase. Specifically,  $k$ -means++ chooses the centroid path one by one to guarantees that the next chosen path is far from the existing centroid paths. We observe that  $k$ -means++ leads to fast

<sup>7</sup>Renormalization can be conducted to convert minimum to maximum.

convergence and a stable objective value, but it makes the objective value higher than in the randomly chosen centroid paths, as it gradually chooses the centroid path far from other existing centroids, and so trajectories in sparse areas tend to be chosen.

## F Cluster Validity Indices (CVI)

There are two types of cluster validation: 1) *external validation* based on previous knowledge about the data (supervised); and 2) internal validation based on the information intrinsic to the data alone [64] (unsupervised). Since there are still no labeled city-wide vehicle trajectories for classification experiments ([54] mainly labeled the dataset from a road intersection in a very small area, which is not exploitable in our case), we mainly consider internal validation as widely used in unsupervised machine learning to evaluate the effectiveness of  $k$ -paths. We use internal measures which consider the intra-cluster *compactness* and inter-cluster *separation* [33, 49]. These methods indicate how closely related objects are in a cluster, and how distinct or well separated a cluster is from other clusters.

Table 8 compares EBD with the state-of-the-art distance measures. In Table 7, we show the formal definition of each CVI. The  $\mu^*$  in *Calinski-Harabasz* is the centroid path of the whole dataset, we run  $k$ -paths by setting  $k = 1$  to get it. The “↑” after each CVI name shows that the higher the value, the better the cluster quality, while “↓” corresponds to worse cluster quality. It is noteworthy that various CVIs have certain limitations in different application scenarios [49], and it remains an open issue to explore which CVIs are proper for trajectory clustering since trajectories are different from Euclidean space data as discussed in the introduction.

We use multiple measures used for point clustering validity such as *Dunn* [24], *Silhouette* [60], *DavidBouldin* [21], *CalinskiHarabasz* [13] evaluated by Liu et al. [49]. The other three basic indices which measure compactness are also used. We ignore LORS here as it has the same value with EBD because of Conjecture 1. Since the distance value varies, we normalize it the maximum distance for all of the distance measures. *k*-center [14] is also used as a baseline.

We can observe that EBD performs well for most cluster quality measures, especially on DavidBouldin, SquaredDistance and AverageDistance. The case study further reinforces our belief that EBD based  $k$ -paths finds frequent paths accurately. A CPEP-based  $k$ -paths method can show the trend clearly, especially for T-drive dataset, while the density based-TRACCLUS algorithm [46] returns a highly concentrated result in the city (Porto), or discrete road segments (T-drive) which are difficult to connect, making trends harder to discover.

## G Zoom-in of Visualized Figures

To clearly observe the locations of clusters, airport, and railway train stations, this section presents a larger version of Figure 15(c) (see Figure 22) and Figure 1(c) (see Figure 21) which could not be presented fully due to the space limitation. We got the information about main railway stations by searching “Beijing railway station” and “Porto railway station” using Google Maps, respectively.

Table 7: Definitions of Internal Cluster Validity Indices.

Measure	Definition	Type
MaximumDiameter↓	$\max_{S_i \in S} \max_{T_x, T_y \in S_i} Dist(T_x, T_y)$	Compactness
SquareDistance↓	$\sqrt{\{\sum_{S_i \in S} \sum_{T \in S_i} Dist^2(T, \mu_i)\} / \{\sum_{S_i \in S} ( S_i  - 1)\}}$	Compactness
AverageDistance↓	$\frac{1}{\sum_{S_i \in S}  S_i ^2} \sum_{S_i \in S} \sum_{T_x, T_y \in S_i} Dist(T_x, T_y)$	Compactness
Silhouette↑	$a(T) = \frac{1}{ S_i } \sum_{T_z \in S_i} Dist(T, T_z), b(T) = \min_{S_j \in S - S_i} \left\{ \frac{1}{ S_j } \sum_{T_z \in S_j} Dist(T, T_z) \right\}$ , $a(T) = \frac{1}{ S_i } \sum_{T_z \in S_i} Dist(T, T_z), b(T) = \min_{S_j \in S - S_i} \left\{ \frac{1}{ S_j } \sum_{T_z \in S_j} Dist(T, T_z) \right\}$	Compactness + Separation
Dunn↑	$\min_{S_i \in S} \min_{S_j \in S} \frac{\min_{T_x \in S_i, T_y \in S_j} Dist(T_x, T_y)}{\max_{T_x, T_y \in S_i} Dist(T_x, T_y)}$	Compactness + Separation
Calinski-Harabasz↑	$\frac{\sum_{S_i \in S} Dist^2(\mu_i, \mu^*)}{\sum_{S_i \in S} \sum_{T \in S_i} Dist^2(T, \mu_i)} \frac{k-1}{ D -k}$	Compactness + Separation
Davies-Bouldin↓	$\frac{1}{k} \sum_{S_i, S_j \in S} \max_{j \neq i} \left( \frac{\frac{1}{ S_i } \sum_{T \in S_i} Dist(T, \mu_i) + \frac{1}{ S_j } \sum_{x \in S_j} Dist(x, \mu_j)}{Dist(\mu_i, \mu_j)} \right)$	Compactness + Separation

Table 8: CVI comparisons of different distance measures-based  $k$ -paths and  $k$ -center.

CVI	Porto							T-drive						
	EBD	EDR	DTW	Fré	Hau	ERP	$k$ -center	EBD	EDR	DTW	Fré	Hau	ERP	$k$ -center
MaximumDiameter	1.00	1.00	0.96	0.89	0.90	2.05	0.86	1.00	1.00	0.95	1.21	1.18	1.21	1.18
SquaredDistance	0.01	0.01	0.01	0.05	0.03	0.05	0.03	0.03	0.04	0.01	0.15	0.15	0.04	0.14
AverageDistance	0.10	0.11	0.06	0.18	0.16	0.18	0.15	0.13	0.15	0.06	0.35	0.33	0.13	0.32
Silhouette	-0.72	-0.46	-0.46	-0.24	-0.19	0.11	-0.18	-0.72	-0.74	-0.42	-0.80	-0.83	-0.07	-0.85
Dunn	0.05	0.05	0.02	0.06	0.07	0.00	0.07	0.02	0.01	0.02	0.15	0.07	0.08	0.07
CalinskiHarabasz	0.95	0.60	1.19	0.61	0.45	0.81	0.45	0.71	0.84	1.43	3.60	4.44	2.40	5.44
DavidBouldin	3.39	3.00	4.17	5.28	3.60	26.30	3.60	7.38	7.39	7.80	4.40	6.48	12.01	7.48

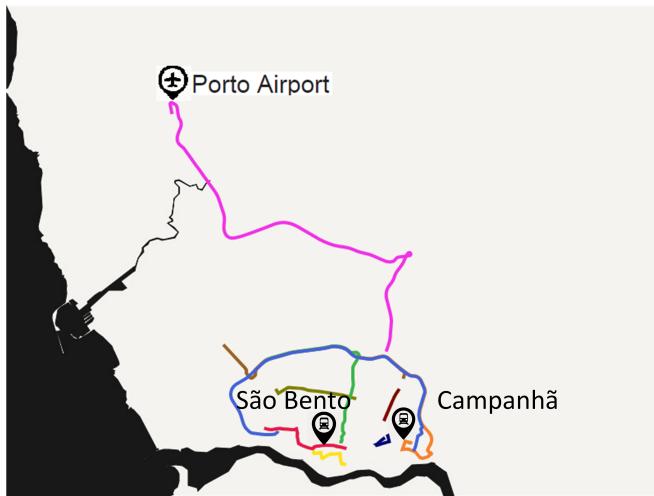


Figure 21: The enlarged version of Figure 1(c).

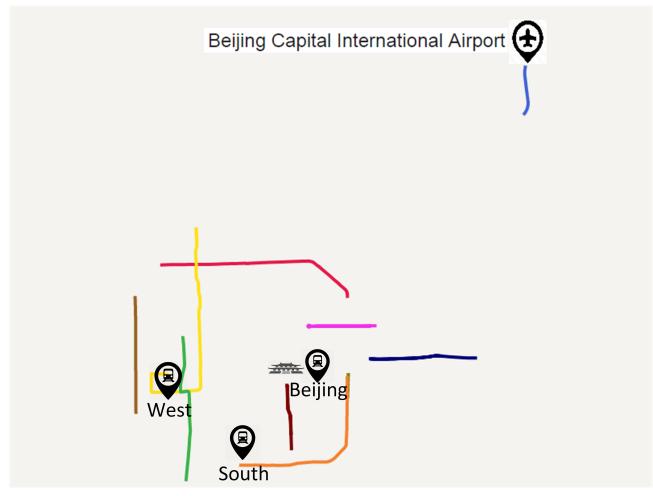


Figure 22: The enlarged version of Figure 15(c).