Fast Large-Scale Trajectory Clustering

Sheng Wang[◇] Zhifeng Bao[†] J. Shane Culpepper[†] Timos Sellis[‡] Xiaolin Qin[§]
[◇]New York University [†]RMIT University
[‡]Swinburne University of Technology
[§]Nanjing University of Aeronautics and Astronautics
[◇]swang@nyu.edu [†]{zhifeng.bao, shane.culpepper}@rmit.edu.au [‡]tsellis@swin.edu.au [§]qinxcs@nuaa.edu.cn

ABSTRACT

In this paper, we study the problem of large-scale trajectory data clustering, ***k*-paths**, which aims to efficiently identify k "representative" paths in a road network. Unlike traditional clustering approaches that require multiple data-dependent hyperparameters, ***k*-paths** can be used for visual exploration in applications such as traffic monitoring, public transit planning, and site selection. By combining map matching with an efficient intermediate representation of trajectories and a novel *edge-based distance* (EBD) measure, we present a scalable clustering method to solve ***k*-paths**. Experiments verify that we can cluster millions of taxi trajectories in less than one minute, achieving improvements of up to two orders of magnitude over state-of-the-art solutions that solve similar trajectory clustering problems.

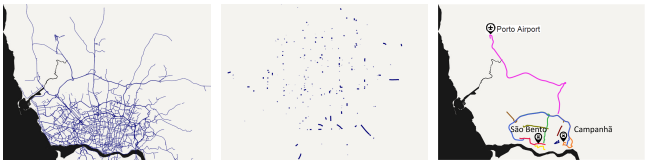
PVLDB Reference Format:
Sheng Wang, Zhifeng Bao, J. Shane Culpepper, Timos Sellis, Xiaolin Qin. Fast Large-Scale Trajectory Clustering. *PVLDB*, 13(1): 29-42, 2019.
DOI: https://doi.org/10.14778/3357377.3357380

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 [50]. Its volume is enormous – a car carrying embedded mobile broadband chips can produce up to 25 gigabytes of data in an hour, including GPS routes [1].

In this paper, we study the problem of large-scale vehicle trajectories clustering. Despite a great deal of progress since 2007 [34] (summarized in Table 1), significant challenges remain in designing scalable algorithms for large-scale trajectory data. Moreover, many algorithms require users to make difficult hyperparameter choices, such as density thresholds in density-based clustering [34, 35], in order to generate candidate line segments. Properly tuning them is challenging even for domain experts (due to various degrees of limited domain knowledge).

Proceedings of the VLDB Endowment, Vol. 13, No. 1
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3357377.3357380



(a) Porto-TRACLUS (b) T-drive-TRACLUS (c) Porto-k-paths

Figure 1: A glimpse of TRACLUS and ***k*-paths**.

Figure 1 visualizes the results of a widely used density based trajectory clustering algorithm, TRACLUS [34], on two taxi trip datasets: Porto [7] and T-drive [56]. TRACLUS 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 TRACLUS, but it is hard to observe any discernible trends; in T-drive only discrete edges which appear disconnected are identified as the clustered result. This suggests 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** [38], where in both problems k is the only parameter required from user. ***k*-paths** is useful in many upstream applications:

- **Scenario 1: Traffic Flow Analysis.** A traffic analyst needs to find the k frequently travelled paths to visually analyze complex transportation networks [9, 24].
- **Scenario 2: Public Transit Planning.** A transport department wants to open k new bus routes to meet growing demands [29] using historical taxi trip records [46, 56].
- **Scenario 3: Site Selection.** A company plans to place k billboards over the busiest routes in a city [57, 58].

Efficient ***k*-means** with Lloyd's algorithm [36] has previously been investigated for large-scale point data [22]. 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) [15] has quadratic complexity. When using EDR, our experiments showed that

*Zhifeng Bao is the corresponding author.
◇Sheng Wang finished this work at RMIT University.

this straightforward approach did not converge after several days when clustering as few as 10,000 trajectories.

Recent studies [48, 55] found that raw trajectories have precision problems derived from GPS errors and sampling rates, and employed *map-matching* [37] of raw trajectories to road network paths when constructing indexes, in order to significantly reduce both storage and computational complexity while achieving better precision when measuring the trajectory similarity. However, the distance computation cost remains quadratic.

In order to overcome the above obstacles when scaling ***k***-paths to larger collections, we propose an efficient clustering method 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 [48]. 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. 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 that is fast to compute (Section 4).
- We propose a clustering method 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 accelerate the pruning in the assignment process, 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 [12, 34] using two real-world datasets. A case-study based on visualization verifies that the most frequent paths can be identified accurately (Section 7).

Complete proofs for lemmas in this paper can be found in our technical report [47]. The source code, curated datasets, and visualization tools are also available [6] 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 [51]. 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* [37] has been shown to be more effective in terms of storage and retrieval performance [48]. Wang et al. [48] modeled road networks as a directed graph, where each road is an edge with a unique ID, and a trajectory can be represented as a sequence of integers.

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

Work	Type	Measure	Data Scale	#Para	Time
k-paths	P	EBD	Million (Vehicle)	1	Minute
[12]	P	Hausdorff	422 (Vehicle)	4	Hours
[51]	P	DTW	4700 (Vessels)	2	NA
[24]	P	Euclidean	372,601 (Vehicle)	2	2497s
[27]	P	Fréchet	2 soccer players	2	700s
[42]	P	ERP	1100 (Vehicle)	4	10s
[50]	P	EDR	100 (Cellular)	3	400s
[9]	D	-	176,000 (Vehicle)	$\|D\|$	NA
[35]	D	-	7000 (Vehicle)	3	100s
[29]	D	Hausdorff	5000 (Vehicle)	6	120s
[34]	D	-	570 (Hurricane)	2	NA
[8]	D	Fréchet	20,000 (Vehicle)	3	50.4hrs

Trajectory Distance Measures. Many distance measures have been proposed for trajectory data. Common measures include Dynamic Time Warping (DTW) [32], Longest Common Sub-sequence (LCSS) [45], Edit Distance on Real sequence (EDR) [15], Hausdorff distance [40], Discrete Fréchet distance [21], and Edit distance with Real Penalty (ERP) [14]. However, all these existing point-based measures have a high complexity (quadratic) and are sensitive to GPS errors, sampling rate, point shifts, or hyperparameter tuning (EDR, LCSS, and ERP all need one parameter) when applied to raw vehicle trajectories. In our recent work [48], experiments verified the robustness of a new distance measure: *the longest overlapped road segments* (LORS). Yuan et al. [55] normalized LORS to measure the distance between two trajectories by considering the length of trajectories.

Trajectory Clustering. As shown in Table 1, existing trajectory clustering methods can be divided into two types [54], Partition-based [12, 24, 27, 30, 42, 51] and Density-based [8, 9, 29, 34, 35]. Note that when the trajectory is stored as points, Ding et al. [20] utilized DBSCAN [23] to cluster trajectory points at a specific timestamp, rather than trajectories as a whole. Based on this categorization, three important observations about current techniques can be made: 1) Existing solutions are only tractable for small datasets such as animals and hurricanes. For example, the most cited trajectory clustering solution [34] clusters only 570 trajectories (so we optimized [34] 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 have a 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, and has a novel distance measure-EBD which is both scalable and effective.

3. DEFINITIONS & PRELIMINARIES

3.1 Trajectory Data Modeling

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

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 paths
O, S	The objective function, clusters
$a'(i)$, $a(i)$	T_i's previously and newly assigned cluster ids
μ_j', μ_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 μ_j
EH, ALH	The edge and accumulated length histograms
$\|e\|$, $\|G_j\|$	The weight of edge e and frequency graph G_j

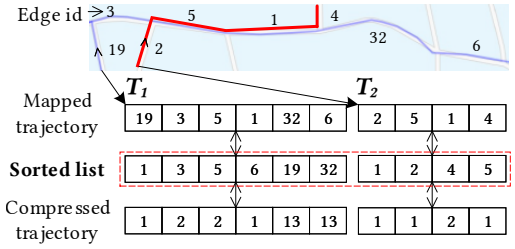


Figure 2: Trajectory data modeling and compression.

Definition 2. ***(Raw Trajectory)*** *A trajectory T of travel length $|T|$ is in the form of $\{p_1, p_2, \ldots, p_m\}$, 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 $e_1 \rightarrow e_2 \rightarrow \ldots \rightarrow e_m$ 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 $T : e_1 \rightarrow e_2 \rightarrow \ldots \rightarrow e_m$.*

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 $\{T_1, T_2, \cdots, T_n\}$, **k-paths** aims to partition the n trajectories into k ($k \leq n$) clusters $S = \{S_1, S_2, \cdots, S_k\}$ to minimize the objective function:*

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

where each cluster S_j has a centroid path μ_j which should be a path in G, and Dist *is the trajectory distance measure.*

The key differences between ***k*-paths** and ***k*-means** are threefold: (1) trajectories can be of varying lengths instead of fixed-length vectors in a Euclidean space; (2) a trajectory distance measure *Dist* for two trajectories must be defined; (3) the centroid path μ_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 [41], 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 [36] 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): $\{\mu_1, \ldots, \mu_k\}$.

2) Assignment. Find the nearest centroid path μ_j for every trajectory T_i in the database, and assign it to the centroid path's affiliated cluster, denoted as $a(i) = j$.[1]

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 $\mathcal{O}(nk) \times \mathcal{O}(dis)$, where n is the total number of trajectories, k is the number of clusters and $\mathcal{O}(dis)$ is the complexity of distance computation which is quadratic when using existing distance measures.
Challenge 2. In the refinement step, the complexity of computing the mean is constant in ***k*-means**, but it does not hold in ***k*-paths**. Ferreira et al. [24] proposed a *fitting field vector* solution, which costs $\mathcal{O}(kn|S(D)|)$, where $S(D)$ denotes the set of line segments that compose the trajectories in the dataset D. This point-based trajectory approach does not scale well in practice based on our experiments (Figure 9). Even though we can extend *k*-medoids [41] to choose an existing trajectory as a new centroid path, a distance matrix with a complexity of $\mathcal{O}(n^2) \times \mathcal{O}(dis)$ must still be computed.

4. A NOVEL DISTANCE MEASURE

Large-scale clustering requires a distance measure that is precise and cheap to compute. The distance measure LORS [48], which is the state-of-the-art in term of precision, can be extended for ***k*-paths** without sacrificing effectiveness.

4.1 Defining LORS

LORS measures the similarity based on the length of overlapped edges while ensuring that matched edges do not violate ordering constraints, which is also known as *local time shifting* [15]. 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 $T_1 = (e_{11}, e_{12}, ..., e_{1m})$ and $T_2 = (e_{21}, e_{22}, ..., e_{2x})$; $|e_{1m}|$ is the travel length of graph edge e_{1m}. $H(T_1) =$

[1] In the rest of the paper, indices i and j always refer to trajectory and cluster indices, respectively.

$(e_{11}, ..., e_{1m-1})$ 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 [15, 52], 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 [47].

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 one 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., $C_o(e_1e_2) = |\{D_s \subset D | \forall T \in D_s : T(e_1) < T(e_2)\}|$, $T(e)$ is the order of e in T.

Based on these observations, we performed additional experiments to compute the successive probability distribution of two datasets 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 in `Porto`. Figure 3 shows that *around* 65% *of edge-pairs* (e_1, e_2) *in* `Porto` *have a stable successive relationship, i.e.,* $\mathbb{P}(e_1, e_2) = 1$*, and more than* 80% *have a probability* $\mathbb{P} > 0.8$.

A visual analysis of the paths on a map shows that a stable successive co-occurrence relationship is mainly because most taxis follow the shortest or fastest path between an origin and destination suggested by the navigation apps in practice (the percentage was reported as **96.8%** [43]). For successive probability less than 100%, it means that a path has a detour with a high chance. For example, `T-drive` has multiple trips in a trajectory without segmentation, so detours are common, and this is confirmed in Figure 15(a)(b). This explains why only 48% of edge-pairs have $\mathbb{P}(e_1, e_2) = 1$, which is much lower than `Porto`. However, the precision of EBD still remains high (91.3%) in `T-drive`. We thus derive the following lemma:

LEMMA 1. *For any two trajectories T_1 and T_2 in the road network G, a* 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 [47], where we prove in two steps: (1) for any two edges e_1 and e_2, $\mathbb{P}(e_1, e_2) = 1$ holds when all the trips are the shortest (fastest) paths in the road network (weighted graph); (2) LORS$(T_1, T_2) = |T_1 \cap T_2|$ holds when $\mathbb{P}(e_1, e_2) = 1$. □

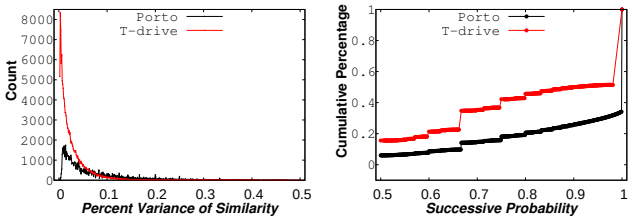


Figure 3: Distribution of variance & successive probability.

Table 3: Time (μs) 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

Based on Lemma 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. 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 *Dist* in Definition 6, we use the trajectory travel length $|T_1|$ and $|T_2|$ to normalize the similarity to a distance value, similar to previous work [15, 55, 45]. Inspired by EDR [15] 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 μ_j when minimizing the objective value in Equation 1. Moreover, EBD obeys the non-negativity, identity of indiscernible, and symmetry.

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:* 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 ($\mathcal{O}(mlogm)$).[2] This is an order of magnitude improvement over unsorted lists, and the fast set intersection is a fundamental problem and being continuously explored [19], which leaves space to improve the efficiency of EBD in the future.

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 distance measures. EBD achieves two orders of magnitude improvement over the other six measures, especially for `T-drive` which stores long trajectories.

[2]The exact cost is $\mathcal{O}(m_1 log(m_2/m_1))$ [18, 19] 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 $\mathcal{O}$(EBD) as EBD's complexity.

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 [59].

EXAMPLE 3. *After sorting the trajectory list incrementally, we can compress it using delta encoding [59] – $T_1 = \{1, 3-1, 5-3, 6-5, 19-6, 32-19\} = \{1, 2, 2, 1, 13, 13\}$, 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 [48]. 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 those commonly used pruning algorithms [25], 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 2. *For any trajectories T_1, T_2, and T_3, we have:*

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

PROOF. This lemma can be proved using a *Venn diagram* [44] with the three trajectories. The detailed proof is in our technical report [47]. □

Any metric index such as a VP-tree [53] or M-tree [16] can be used to index the trajectories and support EBD for fast similarity search. In Section 6.2, we will propose an 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 [11]. Here, we compare two different strategies: 1) randomly choosing k trajectories from dataset; 2) adopting k-means++ [11]. We find that random initialization is sufficient for fast convergence when using EBD for ***k***-paths clustering (details in [47]). Hence, the centroid initialization of ***k***-paths is not further explored here.

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 μ_j, as shown in line 4 to 9 of Algorithm 1 in the first iteration. The overall complexity of each iteration will be $\mathcal{O}(nk) \times \mathcal{O}(\text{EBD})$.

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 and the previous distance computation is a widely adopted method in ***k***-means [22].

Algorithm 1: ***k***-paths (k, D)

Input: k: #clusters, D: dataset.
Output: k centroid paths: $\{\mu_1, \ldots, \mu_k\}$.

```
1  t ← 0, initialize the centroid paths μ = {μ_1, ··· , μ_k};
2  while μ changed or t = 0 do
3      if t = 0 then
4          for every trajectory T_i ∈ D do
5              min ← +∞;
6              for every centroid path μ_j do
7                  lb(i, j) ← EBD(T_i, μ_j);
8                  if lb(i, j) < min then
9                      a(i) ← j, min ← 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 ∈ D do
14             Update ub and lb;
15             if max(lb(i), cb(a'(i))/2) > ub(i) then
16                 T_i stays in current cluster: a(i) ← a'(i);
17             else
18                 min ← +∞;
19                 for every centroid path μ_j do
20                     if ub(i) > lb(i, j) then
21                         lb(i, j) ← EBD(T_i, μ_j);
22                         if lb(i, j) < min then
23                             a(i) ← j, min ← lb(i, j);
24             if a'(i) ≠ a(i) then
25                 HistogramUpdate(a(i), EH, ALH, T_i);
26     for every centroid path μ_j do
27         Compute O_j (Equation 10) and update μ_j;
28     t ← t + 1;
29 return {μ_1, ..., μ_k};
```

We now show how our bounding method solves 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 μ_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

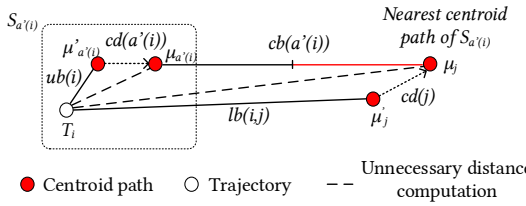


Figure 4: An example of lower bound and pruning.

paths $\mu_j^{'}$ for all $k-1$ clusters, i.e., $lb(i,j) = \mathsf{EBD}(T_i, \mu_j^{'})$, where $j \in [1,k]$ and $j \neq a^{'}(i)$, and (2) an upper bound distance $ub(i) = \mathsf{EBD}(T_i, \mu_{a^{'}(i)}^{'})$. After each refinement, the distance between the current centroid μ_j and the previous centroid $\mu_j^{'}$ in cluster S_j will be computed as the *centroid drift* $cd(j) = \mathsf{EBD}(\mu_j^{'}, \mu_j)$. 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, T_i cannot be assigned to the centroid path μ_j $(a(i) \neq j)$ if $ub(i) < lb(i,j)$, denoted as:

$$lb(i,j) > ub(i) : a(i) \neq j \tag{6}$$

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:

$$lb(i) > ub(i) : a(i) = a^{'}(i) \tag{7}$$

Note that $lb(i,j)$ will be updated as $\mathsf{EBD}(T_i, \mu_j)$ if it is computed during the assignment (line 21); otherwise, we keep the current bound for next iteration. The same applies to $ub(i)$ if $\mathsf{EBD}(T_i, \mu_{a^{'}(i)})$ 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, which can be completed immediately as k is always small. Also, we store the minimum distance $cb(a^{'}(i)) = \min_{j \neq a^{'}(i)} \mathsf{EBD}(\mu_{a^{'}(i)}, \mu_j)$ 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 $S_{a^{'}(i)}$:

$$\begin{aligned} \frac{\mathsf{EBD}(\mu_{a^{'}(i)}, \mu_j)}{2} &> ub(i) : a(i) \neq j \\ \frac{cb(a^{'}(i))}{2} &> ub(i) : a(i) = a^{'}(i) \end{aligned} \tag{8}$$

Then, we combine two bounds to induce further pruning:

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

Example 4. *In Figure 4, T_i was assigned to $S_{a^{'}(i)}$ in previous iteration.*[3] *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 $\mathsf{EBD}(T_i, \mu_{a^{'}(i)})$ and $\mathsf{EBD}(T_i, \mu_j)$, we compute the upper bound of $\mathsf{EBD}(T_i, \mu_{a^{'}(i)})$ as $ub(i) = ub(i) + cd(a^{'}(i))$, and the lower bound of $\mathsf{EBD}(T_i, \mu_j)$ as $lb(i,j) = |lb(i,j) - cd(j)|$ using the triangle inequality. If $lb(i,j) > ub(i)$, then $\mathsf{EBD}(T_i, \mu_j) > \mathsf{EBD}(T_i, \mu_{a^{'}(i)})$, which means T_i is closer to $\mu_{a^{'}(i)}$ than μ_j, and T_i can stay in cluster $S_{a^{'}(i)}$. Similarly, another bound $\frac{cb(a^{'}(i))}{2}$ can be used to compare with $ub(i)$ for pruning.*

[3]For a clearer observation of the pruning, the trajectory and EBD distance are simply drawn as point and line in a metric space.

5.3 Centroid Path Refinement

Similar to k-`medoids` [41], 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:

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

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 a sum checking over all the trajectories in each cluster is performed, and the trajectory with minimum distance as the new centroid path is chosen. The above baseline has a time complexity of $\mathcal{O}(|S_j|^2) \times \mathcal{O}(\mathsf{EBD})$ 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:

$$\begin{aligned} O_j &= \arg\min_{\mu_j \in S_j} \Big(\sum_{T \in S_j} max(|T|, |\mu_j|) - \sum_{e \in \mu_j} \|e\| \Big) \\ &= \arg\min_{\mu_j \in S_j} \Big(\sum_{T \in S_j} |T| + \sum_{T \in S_j^{'}} (|\mu_j| - |T|) - \sum_{e \in \mu_j} \|e\| \Big) \\ &= \arg\min_{\mu_j \in S_j} \big(\|G_j\| + \sum_{T \in S_j^{'}} (|\mu_j| - |T|) - \sum_{e \in \mu_j} \|e\| \big) \end{aligned} \tag{11}$$

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, $|e|$ (the length of e), and $S_j^{'}$ 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 *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*. Through the above transformation, trajectories with frequent edges are selected as the centroid paths. This further verifies that the paths returned from EBD-based ***k*-paths** are frequency-based representatives for the whole dataset.

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 each iteration, instead we 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 and there will be fewer updates to the histogram. For all of the clusters, we also maintain a global edge histogram

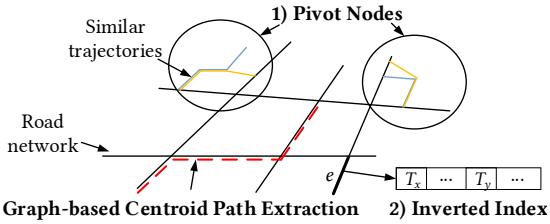


Figure 5: An overview of indexing framework PIG.

(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:

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

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

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

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 $\mathcal{O}(|S_j|^2) \times \mathcal{O}(\mathsf{EBD})$ 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:

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

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

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 Discussion

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. Scanning every trajectory does not scale in large collections, and so indexing techniques can be used to minimize trajectory data processing costs.

Moreover, choosing an existing trajectory as the 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 have a more robust refinement process, 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 μ_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 range query, path query, and top-k similarity search [48] if it is used for road network trajectories, thus allowing us to interactively explore the trajectories in a specific range or path efficiently.

6.2 Pivot-table for Metric Features

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. [31] 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 the index can greatly improve performance for ***k***-means, especially in low dimensional space [28]. 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 [13]. 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 each 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 $dist(q, T_p) - r > min_{dist}(q)$, where $min_{dist}(q)$ is the current minimum distance. The assignment can be accelerated by pruning a group of trajectories.

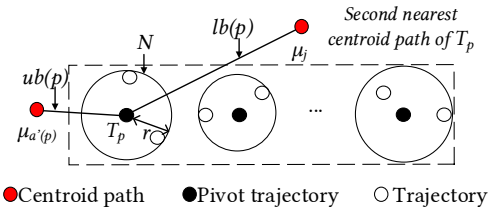


Figure 6: Batch assignment with Pivot-table.

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:

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

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

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

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

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 μ_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. μ_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 μ_j), i.e., $lb(p) - r > ub(p) + r$.*

Algorithm 2: PivotTableConstruction(k, D)

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

```
1  PT ← ∅, Q.push(D);
2  while Q is not empty do
3  |  D_s ← Q.poll();
4  |  S ← k-paths(k, D_s);
5  |  for every cluster S_j ∈ S do
6  |  |  if |S_j| ≤ k then
7  |  |  |  r ← GetRadius(S_j, μ_j);
8  |  |  |  PT.add(μ_j, r, S_j);
9  |  |  else
10 |  |  |  Q.push(S_j);
11 return PT;
```

To group all trajectories into nodes, an M-tree [16] 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 μ_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.

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 μ_j^G in G to minimize the EBD with all the trajectories in the cluster S_j, which can be formulated as:*

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

where $\mu_j^G \in G_j$ denotes that μ_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 of 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) [26, 39].

Lemma 3. *The CPEP which finds a path μ_j^G in graph G_j for O_j^G is NP-hard.*

Proof. We can convert Equation 19 as follows:

$$\begin{aligned} O_j^G &= \arg\min_{\mu_j^G \in G_j} \left(\|G_j\| + ALH_j[|\mu_j^G|] - \max \sum_{e \in \mu_j^G} EH_j(e)\right) \\ &= \arg\min_{|\mu_j^G|} \left(\|G_j\| + ALH_j[|\mu_j^G|] - kTSP(|\mu_j^G|, G_j)\right) \end{aligned} \tag{20}$$

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) [26, 39] (*Given a weighted graph, find a path of minimum*[4] *weight that passes through any k vertices*, where we can relax our problem by setting the length of each edge e as 1 similar to Example 2, i.e., $|e| = 1$). Hence, CPEP is NP-hard. □

Note that a bounded approximation ratio for k-TSP can be obtained only under the assumption that the edge-lengths satisfy the triangle inequality [26, 10], 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.*

Algorithm 3: RefineCPEP$(G, \mu_j^{'})$

Input: G: graph, $\mu_j^{'}$: previous centroid path.
Output: the centroid path μ_j.

```
1  PQ ← ∅, B ← ∅, min ← O_j^G(μ_j'), it ← 0, μ_j ← μ_j';
2  for each edge e ∈ EH_j do
3      if min > LB(e) then
4          PQ.push(e, LB(e));
5  while PQ.IsNotEmpty() do
6      (ps, LB(ps)) ← PQ.poll();
7      if LB(ps) ≥ min or it ≤ it_max then
8          break;
9      for each neighbor edge e of ps do
10         if EH_j.contains(e) and e ∉ ps then
11             ps ← ps + e;
12             Compute O_j^G(ps) by Equation 19;
13             if O_j^G(ps) < min then
14                 min ← O_j^G(ps), μ_j ← ps;
15             Compute LB(ps) by Equation 21;
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 ← it + 1;
21 return μ_j;
```

The CPEP can be solved by setting different $|\mu_j^G| \in [l_{min}, l_{max}]$ for k-TSP,[5] 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 μ_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 paths are grown by appending neighbor edges using a breadth-first search. We call this process as *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*).

[4]Renormalization can be conducted to convert minimum to maximum.

[5]Here, k is the number of edges, which is a different concept from the k in k-paths.

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 [17]. We do not insert this kind of path into the priority queue. Moreover, a consistent path should be only expanded to a start edge and an end edge, and the connectivity of these two edges (be, ee) can be retrieved from the graph G. Further, we check whether there is a trajectory crossing e, i.e., $EH_j.contains(e)$.

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.

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:

$$LB(ps) = \|G_j\| - \sum_{e \in ps} EH_j(e) + \min_{l=|ps|+1}^{l_{max}} \left(ALH_j[l] - \phi(EH_j, l)\right) \quad (21)$$

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

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 21), 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 wish to conduct experiments to show the efficiency of EBD based ***k*-paths** over state-of-the-art [12, 24], scalability of the PIG index proposed for ***k*-paths**, the effectiveness of EBD over state-of-the-art [12, 34, 24], as well as the impact of varying k.

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

(a) Porto (b) T-drive

Figure 7: Road network overview of two collections.

Datasets. 1) Porto [7]: 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 [56]: trips for 30,000 taxis in Beijing over three months. Porto and T-drive are mapped into the 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.

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 [34] which is density-based, and the state-of-the-art k-center for point-based trajectories [12] which is partition-based. We also use LORS and five other existing trajectory distance measures widely used for partition-based clustering as shown in Table 1, and integrate them with ***k*-paths**. Specifically, DTW [51], Discrete Fréchet Distance (Fré for short) [8], and EDR [50], Hausdorff (Hau for short) [12], ERP [42] are compared with ***k*-paths** and EBD, where the starting vertices of edges embody the point-based trajectory. Running time, a 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.** In Figure 9, we first compare the single-iteration running time of EBD-based ***k*-paths** with that of other distance measures based ***k*-paths** for Porto (see [47] for the performance on T-drive).

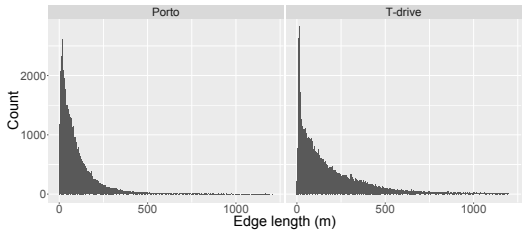


Figure 8: Edge length histograms of two datasets.

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 omit EDR as it has a similar running time to LORS. Both use a similar dynamic programming approach, and the key difference is that EDR is point-based while LORS is based on edges. All other distance measures use assignments and refinements based on a distance matrix. We also show the number of iterations for all distance measures (Note that EBD and LORS overlap as they produce a similar trajectory distance on the collections, and require the same number of iterations).

***k*-paths with EBD.** The histograms in Figure 9 show the time spent on assignment and refinement respectively, and compare five 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 [12] which adopts the Hausdorff distance measure [40] and uses an alternative partition-based clustering framework. k-center tries to bound all points in k circles, and minimize the sum of the radius; **5)**: VFKM [24] (Vector Field k-means) which uses *vector fields* to induce a notion of similarity between trajectories, define and represent each cluster. 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 process 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 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.

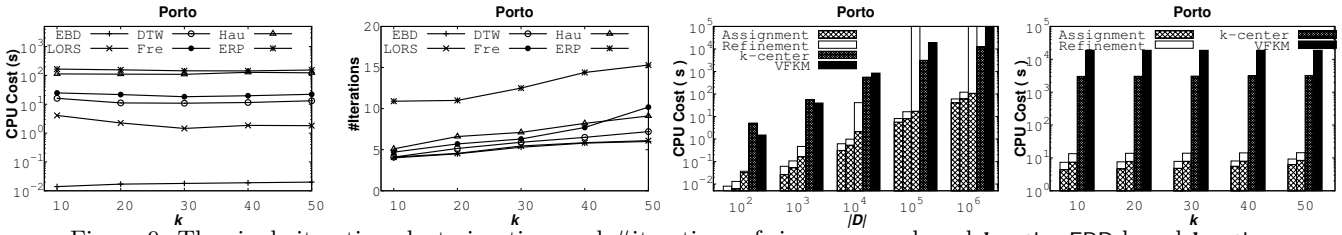


Figure 9: The single iteration clustering time and #iterations of six measures-based k-paths, EBD-based k-paths.

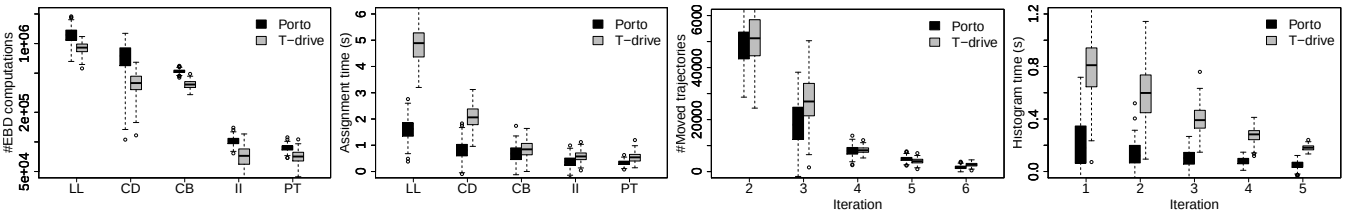


Figure 10: Assignment Breakdown: #EBD computation and time, #moved trajectories and histogram update time.

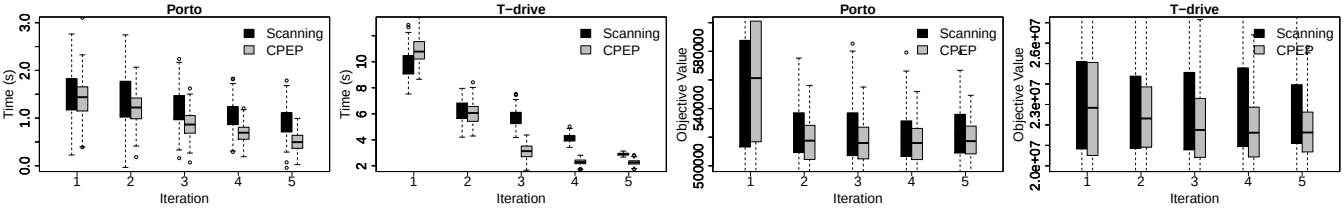


Figure 11: Refinement Breakdown: time and objective value in each iteration for scanning and CPEP.

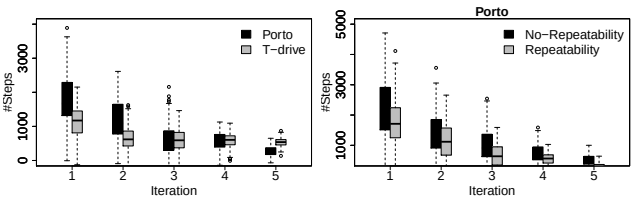


Figure 12: CPEP Breakdown: #steps in each iteration.

Table 5: Construction time (sec), 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	-

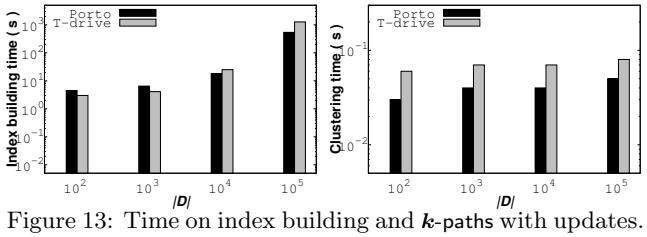


Figure 13: Time on index building and k-paths with updates.

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 methods–k-center [12] and VFKM [24] by roughly two orders of magnitude. Specifically, k-paths requires fewer distance computations (more than 80% are pruned) by using an 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 increases slightly. Increasing k leads to a small rise in the number of iterations and running time.

7.2 Scalability Study of Indexing

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. We also perform a scalability test on pivot-table construction by increasing the size of the data in both datasets in Figure 13.

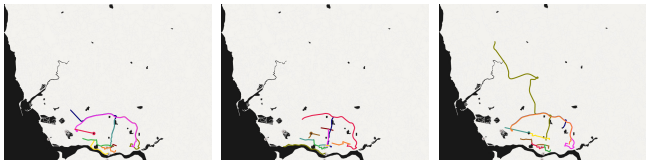
k-paths with Updates. When we insert a new trajectory into the dataset, efficiently updating the centroid path is crucial. We can assign this new trajectory to k centroid paths using the lower bound technique, update the corresponding cluster's edge and length histograms, and then we can run CPEP and 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 [59]. 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

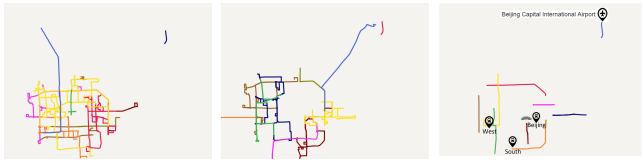
7.3.1 Case Study by Visualization

As shown in Figure 14, 15, 16, and Figure 1 which we introduced previously, we cluster the taxi datasets to help plan



(a) LORS-based k-paths (b) k-center (c) EBD-based k-paths

Figure 14: Case studies on Porto.



(a) k-center (b) EBD-based k-paths (c) CPEP-based k-paths

Figure 15: Case studies on T-drive.

new bus routes. We compare six different methods: 1) TRACLUS; 2) LORS-based; 3) k-center [12]; 4) EBD-based without CPEP; 5) EBD-based with CPEP; 6) VFKM [24]. Figure 1(a)(b) presents the density-based method TRACLUS [34]. The remaining five are partition-based methods and are presented in Figure 14, 15, 16(a)(b), and 1(c), and the output is k paths, each shown with a different color. Note that we omit LORS-based for T-drive due to space limitations.

Since the original TRACLUS algorithm [34] needs to process every trajectory to generate line segments, and is not scalable to cluster million-scale datasets, we have optimized TRACLUS using our data modeling method. TRACLUS 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. [34] set it as the average frequency by default), which can be done with our edge histogram. Figure 16(c) shows the ten most frequent edges in Porto. We find that they gather at the center of Porto and are too short to be particularly informative.

In Figures 1(c) and 15(c) we highlight the locations of the airport and main railway stations in each city. For Beijing, T-drive has three main railway stations, i.e., Beijing station, Beijing West, and Beijing South; Porto has two main railway stations, São Bento and Campanhã. We can observe that our cluster paths are often closely aligned with these prominent locations (see [47] for more detailed figures).

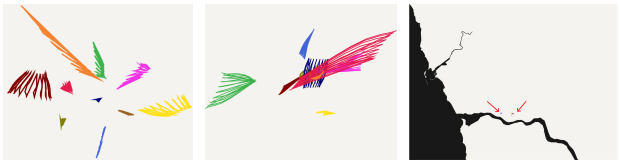
7.3.2 Choice of k

As k is the only hyperparameter required for k-paths, we also explore how to estimate the best k for each dataset. Using the elbow method [33] 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 17, we increase k and observe changes to the objective values produced.

Observation 3. *Firstly, Figure 1, 14, and 15 both show that the returned centroid paths for EBD based k-paths cover the path beginning from or ending at the airport while the other two methods (LORS and k-center) do not. Since VFKM [24] generates vector fields instead of using existing trajectories on the road network, the results are not real paths in a road network. LORS performs poorly as we can only use it to cluster 1000 trajectories. This is consistent with our intuition that taxis are one of the primary modes of transportation around the airports and railway stations, and provides additional qualitative evidence of the quality of the results being returned by our approach.*

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 17 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 $k = 13$.



(a) VFKM on Porto (b) VFKM on T-drive (c) Ten most frequent edges

Figure 16: Case studies of VFKM and frequent edges.

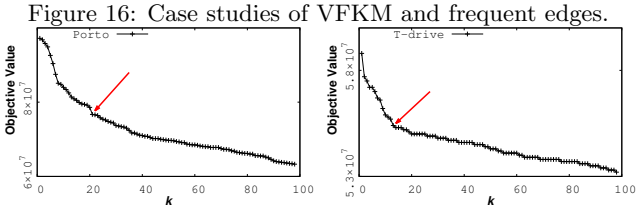


Figure 17: Finding the elbow of the objective value.

Remarks. More effectiveness studies can be explored using our demo system[6] with interactive visualizations (see Scenario 2 of [49] for instructions, the response time may vary due to network delay).

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 using a road network, and propose a distance measure called EBD, which reduces the time complexity of distance computation and obeys the triangle inequality. Based on EBD, effective lower bound pruning 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 uses an inverted index to avoid unnecessary distance computations, and traverses a graph to find k centroid paths. In the future, we hope to further explore the distributed k-paths problem as well as k-paths on other types of trajectory data.

Acknowlegements. This work was partially supported by ARC DP170102726, DP180102050, DP170102231, and NSFC 61728204, 91646204. Zhifeng Bao and Shane Culpepper are recipients of the Google Faculty Research Award.

[6] https://sites.google.com/site/shengwangcs/torch

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] Repository of k-paths. https://github.com/tgbnhy/k-paths-clustering.

[7] Taxi Service Trajectory Prediction Challenge 2015. http://www.geolink.pt/ecmlpkdd2015-challenge/.

[8] 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.

[9] 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.

[10] S. Arora and G. Karakostas. A 2+ ϵ approximation algorithm for the k-MST problem. *Math. Program*, 107(3):491–504, 2006.

[11] D. Arthur and S. Vassilvitskii. K-Means++: The advantages of careful seeding. In *SODA*, pages 1027–1025, 2007.

[12] T.-h. H. Chan, A. Guerquin, and M. Sozio. Fully dynamic k-center clustering. In *WWW*, pages 579–587, 2018.

[13] L. Chen, Y. Gao, B. Zheng, C. S. Jensen, H. Yang, and K. Yang. Pivot-based metric indexing. *PVLDB*, 10(10):1058–1069, 2017.

[14] L. Chen and R. Ng. On the marriage of Lp-norms and edit distance. In *VLDB*, pages 792–803, 2004.

[15] L. Chen, M. T. Özsu, and V. Oria. Robust and fast similarity search for moving object trajectories. In *SIGMOD*, pages 491–502, 2005.

[16] 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.

[17] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to algorithms*. MIT press, 2009.

[18] J. S. Culpepper and A. Moffat. Efficient set intersection for inverted indexing. *ACM Transactions on Information Systems*, 29(1):1–25, 2010.

[19] B. Ding and A. C. König. Fast set intersection in memory. *PVLDB*, 4(4):255–266, 2011.

[20] X. Ding, L. Chen, Y. Gao, C. S. Jensen, and H. Bao. UlTraMan : A unified platform for big trajectory data management and analytics. *PVLDB*, 11(7):787–799, 2018.

[21] T. Eiter and H. Mannila. Computing discrete Fréchet distance. Technical report, 1994.

[22] C. Elkan. Using the triangle inequality to accelerate k-means. In *ICML*, pages 147–153, 2003.

[23] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *SIGKDD*, pages 226–231, 1996.

[24] N. Ferreira, J. T. Klosowski, 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.

[25] V. Ganti, R. Ramakrishnan, J. Gehrke, A. Powell, and J. French. Clustering large datasets in arbitrary metric spaces. In *ICDE*, pages 502–511, 1999.

[26] N. Garg. Saving an Epsilon: a 2-approximation for the k-MST problem in graphs. In *STOC*, pages 396–402, 2005.

[27] J. Gudmundsson and N. Valladares. A GPU approach to subtrajectory clustering using the Fréchet distance. In *SIGSPATIAL*, pages 259–268, 2012.

[28] G. Hamerly. Making k-means even faster. In *SDM*, pages 130–140, 2010.

[29] 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.

[30] 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.

[31] 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.

[32] E. Keogh. Exact indexing of dynamic time warping. In *VLDB*, pages 406–417, 2002.

[33] 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.

[34] J.-g. Lee, J. Han, and K.-Y. Whang. Trajectory clustering: A partition-and-group framework. In *SIGMOD*, pages 593–604, 2007.

[35] Z. Li, J.-G. Lee, X. Li, and J. Han. Incremental clustering for trajectories. In *DASFAA*, pages 32–46, 2010.

[36] S. P. Lloyd. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28(2):129–137, 1982.

[37] Y. Lou, C. Zhang, Y. Zheng, X. Xie, W. Wang, and Y. Huang. Map-matching for low-sampling-rate GPS trajectories. In *GIS*, pages 352–361, 2009.

[38] 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.

[39] V. Nagarajan and R. Ravi. The directed orienteering problem. *Algorithmica*, 60(4):1017–1030, 2011.

[40] S. Nutanong, E. H. Jacox, and H. Samet. An incremental Hausdorff distance calculation algorithm. *PVLDB*, 4(8):506–517, 2011.

[41] 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.

[42] 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.

[43] 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, 2006.

[44] F. Ruskey and M. Weston. A survey of Venn diagrams. *Electronic Journal of Combinatorics*, 4:3, 1997.

[45] M. Vlachos, G. Kollios, and D. Gunopulos. Discovering similar multidimensional trajectories. In *ICDE*, pages 673–684, 2002.

[46] S. Wang, Z. Bao, J. S. Culpepper, T. Sellis, and G. Cong. Reverse k nearest neighbor search over trajectories. *IEEE Transactions on Knowledge and Data Engineering*, 30(4):757 – 771, 2018.

[47] S. Wang, Z. Bao, J. S. Culpepper, T. Sellis, and X. Qin. Fast large-scale trajectory clustering. `https://t4research.github.io/k-paths-tr.pdf`.

[48] 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.

[49] S. Wang, Y. Shen, Z. Bao, and X. Qin. Intelligent traffic analytics: from monitoring to controlling. In *WSDM*, pages 778–781, 2019.

[50] 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.

[51] D. Yao, C. Zhang, Z. Zhu, J. Huang, and J. Bi. Trajectory clustering via deep representation learning. In *IJCNN*, pages 3880–3887, 2017.

[52] B.-k. Yi and F. Park. Effcient retrieval of similar time sequences under time warping. In *ICDE*, pages 201–208, 1997.

[53] P. N. Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. In *SODA*, pages 311–321, 1993.

[54] 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.

[55] H. Yuan and G. Li. Distributed in-memory trajectory similarity search and join on road network. In *ICDE*, pages 1262–1273, 2019.

[56] 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.

[57] P. Zhang, Z. Bao, Y. Li, G. Li, Y. Zhang, and Z. Peng. Trajectory-driven influential billboard placement. In *KDD*, pages 2748–2757, 2018.

[58] Y. Zhang, Y. Li, Z. Bao, S. Mo, and P. Zhang. Optimizing impression counts for outdoor advertising. In *KDD*, pages 1205–1215, 2019.

[59] J. Zobel and A. Moffat. Inverted files for text search engines. *ACM Computing Surveys*, 38(2):1–56, 2006.