Fast Large-Scale Trajectory Clustering

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

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

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

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

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

ABSTRACT

Clustering of large-scale trajectory data is an important and challenging problem, with many useful applications such as public transit planning, site selection, and data visualization. However, manipulating trajectories in large collections is often expensive and unnecessary. In this paper, we propose the k-paths problem, where the goal is to efficiently identify a subset of "representative" road network paths that can be used for improving visualization and manipulation of trajectory data. By combining map matching with an efficient intermediate representation of trajectories, we are able to solve k-paths using a novel *edge-based distance* (EBD) measure, which has high precision and low complexity, and a parameterless clustering algorithm. Experiments verify that we can cluster millions of taxi trajectories in around one minute using our new approach, achieving improvements of up to two orders of magnitude over state-of-the-art solutions for the trajectory clustering problem.

PVLDB Reference Format:
Sheng Wang, Zhifeng Bao, J. Shane Culpepper, Timos Sellis, Xiaolin Qin. k-paths: Fast Large-Scale Trajectory Clustering. *PVLDB*, 12(xxx): xxxx-yyyy, 2019.
DOI: https://doi.org/10.14778/xxxxxxx.xxxxxxx

1. INTRODUCTION

Trajectory data is now ubiquitous and captured from a diverse range of resources, ranging from Global Positioning System (GPS) devices to cameras and radio frequency identification (RFID) readers [66]. For example, a car carrying embedded mobile broadband chips can produce up to 25 gigabytes of cloud-based data in an hour, including GPS routes[1]. The demand for analyzing trajectory data for the purposes of urban-level traffic control, route planning and facility deployment has exceeded all expectations in the last decade, and now realtime processing constraints are routinely imposed on these and many other related applications [72].

In this paper, we study the fundamental problem of large-scale vehicle trajectories clustering. Despite a great deal of progress on the problem since the earliest studies on trajectory clustering [43, 35] (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 [43, 44], in order to generate candidate line segments. Tuning these sensitive parameters is challenging even for domain experts (due to different degrees of (limited) domain knowledge on the data) [28, 65]. Making complex spatial algorithms simpler to use is a growing priority as non-experts increasingly become reliant on the analytical tools being provided [38].

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 for k groups. k-paths is reminiscent of the classical k-means clustering algorithm [50], where in both problems k is the only parameter required. k-paths can be applied in many analytical applications, such as:

- **Scenario 1: Traffic Flow Analysis.** A traffic analyst needs to find the k frequent paths [49] to visually analyze complex transportation networks [2, 23].
- **Scenario 2: Bus Route Planing.** A transportation department wants to open k new bus routes to meet growing demands [31, 40] using historical vehicle trip records [70].
- **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 [71, 45].

Efficient k-means clustering with Lloyd's algorithm [47] has previously been investigated for large-scale data [19, 54]. By choosing k objects as the initial centroids, Lloyd's algorithm iteratively assigns each object in the dataset to the nearest centroid, and refines the new centroid using the mean value of all the objects in each cluster. k-paths can be answered by extending the *assignment and refinement* method. However, scaling this approach is challenging since the assignment requires $\mathcal{O}(nkt)$ trajectory distance computations, where n is the number of trajectories and t is the number of iterations. Meanwhile, a simple refinement of k-means cannot

[1] `https://perma.cc/BPM2-QNW4`

Proceedings of the VLDB Endowment, Vol. 12, No. xxx
ISSN 2150-8097.
DOI: https://doi.org/10.14778/xxxxxxx.xxxxxxx

be used for k-paths because the mean distance cannot accurately represent a real trajectory path (a thorough analysis can be found in Section 3.3).

To overcome the above challenges and answer k-paths efficiently, we propose a parameterless processing framework with the following five core components:

1. Lightweight Trajectory Modeling: Instead of expensive point-based trajectory clustering [43, 8], we transform the raw trajectories into road network paths as the vehicle trajectory data is network-constrained [42]. Then we store trajectories as a list of road segment IDs, which can be further sorted to enable a more efficient trajectory distance computation and to reduce space requirements using delta compression [73].

2. Edge-based Distance Measure: Existing trajectory distance measures, such as Edit Distance on Real Sequences (EDR) [11, 52], which is arguably one of the most commonly used and efficient measures, has quadratic complexity. This makes trajectory clustering more expensive than k-means in Euclidean space where the distance computation is linear. Moreover, EDR needs to specify a range threshold when computing the distance. Therefore, we propose an edge-based distance measure called EBD designed specifically for efficient clustering. EBD computes the distance between two trajectories based on the intersecting road segments and has precision comparable to EDR, it is also parameter-free.

3. Fast Assignment: All of the current pruning strategies such as "bound and index" for k-means in Euclidean space cannot be used for k-paths as most existing trajectory distance measures do not obey the triangle inequality (non-metric). After proving that EBD satisfies the triangle inequality, we employ a lower bound technique to prune the computational space. To overcome the shortcoming of scanning every trajectory iteratively in the baseline, we further propose an indexing framework to accelerate the assignments.

4. Robust Refinement: To update the centroid path efficiently, instead of building a quadratic time distance matrix to find the trajectory which can minimize the objective value, we present a linear-time approach that exploits the *length histogram* and the *edge histogram*. We further extract the centroid path by traversing the road network graph, which is independent of the number of trajectories in the cluster and can further improve the quality of outputs in terms of visualization. Since the *centroid path extraction problem* (CPEP) is known to be NP-hard, an efficient greedy algorithm is proposed to solve the problem.

5. Comprehensive Evaluation: 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 [43, 8] using three real-world datasets. A case-study based on visualization methods verifies that the most frequent paths can be identified using our EBD based k-paths solution efficiently and accurately.

To summarize, we have made the following contributions:

- We define a fundamental trajectory clustering problem k-paths based on a lightweight trajectory modeling method (Section 3), combined with a new distance measure EBD designed for efficient k-paths clustering (Section 4).

- We propose a parameterless framework with low complexity for k-paths based on Lloyd's algorithm, coupled with lower bounds based assignment and histograms based refinement to improve clustering performance (Section 5).

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
[8]		Hausdorff	422 (Vehicle)	4	Hours
[67]		DTW	4,700 (Vessels)	2	NA
[23]		Euclidean	372,601 (Vehicle)	2	2,497s
[26]	P	Fréchet	2 soccer players	2	700s
[60]		ERP	1100 (Vehicle)	4	10s
[66]		EDR	100 (Cellular)	3	400s
[33]		CATS	5,000 (Vehicle)	4	100,000s
[2]		-	176,000 (Vehicle)	$\lvert D \rvert$	NA
[44]		-	7,000 (Vehicle)	3	100s
[31]	D	Hausdorff	5000 (Vehicle)	6	120s
[43]		-	570 (Hurricane)	2	NA
[1]		Fréchet	20,000 (Vehicle)	3	50.39 hours

- We design an indexing framework called PIG to further prune trajectories during assignment, and transform the refinement to a graph traversal problem – CPEP. PIG is also capable of supporting streaming clustering of frequently-updated traffic records from road cameras and GPS (Section 6).

- We use three real-world datasets to verify our proposed approaches. The experimental results show that our algorithm is fast and accurate (Section 7).

All the proofs of theorems and lemmas, and the time complexity analysis of our solutions are in technical report.

2. RELATED WORK

In this section, we review the existing studies related to k-paths, including trajectory modeling, distance measures and existing representative clustering work.

Trajectory Modeling. A set of coordinates denoted by two floats is a traditional representation of trajectories for storage, search and analytics. Raw data representations are easy to work with, but are not space efficient. To further reduce space, converting the raw data to a vector [67] of the same length can simplify processing. This is a relatively simple conversion, but 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* [48] has been shown to be more effective in terms of storage [62] and retrieval performance [64]. The road network is modeled 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 in recent years (more than 10). Common measures include Dynamic Time Warping (DTW) [37], Edit Distance on real Sequence (EDR) (an extension of a similarity measure–Longest Common Subsequence (LCSS) [63], they all need to set a range threshold to find matching points and further compute the edit distance), Hausdorff distance [56], Discrete Fréchet distance [21], and Edit distance with Real Penalty (ERP) [10]. However, they always try to find the best matching point without breaking ordering constraints, thus requiring dynamic programming solutions with a complexity of $\mathcal{O}(n^2)$ (see Table 3). Moreover, all the existing measures are non-metric except for ERP, as they do not obey the triangle inequality. However, ERP has a complexity of $\mathcal{O}(n^2)$ despite being metric, and is therefore not useful for problems such as k-paths in large collections.

Trajectory Clustering. As shown in Table 1, existing trajectory clustering methods can be divided into two types [69], Partition-based: [8, 23, 67, 60, 26, 33, 32] and Density-based: [43, 31, 44, 1, 2, 27]. Note that partition-based clustering is popular not only for vehicle trajectory data, but also for time series data, such as k-Shape [57, 34]. Based on this categorization, three important observations about current techniques can be made: 1) Existing solutions are tractable for small datasets such as animals and hurricanes. For example, the mostly cited trajectory clustering solution [43] clusters only 570 trajectories; 2) Clustering is done on raw trajectory composed of points, and the distance measure computations need quadratic complexity; 3) Most of the solutions have at least two threshold parameters which are highly sensitive to the dataset, making them difficult to reproduce or use in practice (therefore 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 $p = \{lat, lng, ts\}$ contains the latitude lat, the longitude lng and the time-stamp ts.*

Definition 2. ***(Raw Trajectory)*** *A trajectory T of length $|T|$ is in the form of $\{p_1, p_2, \ldots, p_{|T|}\}$, 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 $e_1 \to e_2 \to \ldots \to e_{|P|}$ in G. The travel distance of the path is defined as the sum of length of all edges.*

Since vehicles are moving over the road network, the raw trajectories of vehicles can be mapped as a road network path. This is commonly referred to as *map matching* [48, 55].

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 $T : e_1 \to e_2 \to \ldots \to e_{|T|}$.*

Recent work [64, 31] have shown that mapping trajectories to road network can significantly reduce both the storage complexity and distance computation complexity, meanwhile achieving a better effectiveness for similarity search. Since distance computation is a frequently involved operator in clustering, we choose to adopt the same map matching before applying the clustering algorithm. Example 1 shows a case of converting a raw trajectory to an *edge id list*. In the rest of this paper, we will simply use *trajectory* to represent the *mapped trajectory*. Moreover, the frequent notations are summarized in Table 2.

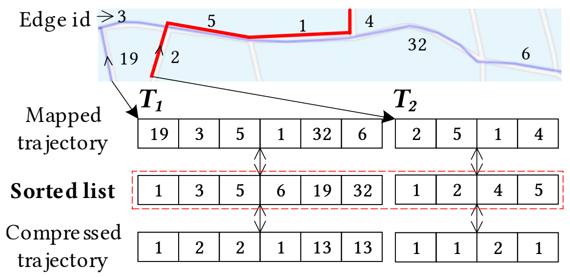


Figure 1: Trajectory data modeling and compression.

Table 2: Summary of Notations.

Symbols	Description
T_i, D	A trajectory, and the dataset
$\|T_i\|$	The number of edges in 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 previous 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

Example 1. *As shown in Figure 1, we have two trajectories T_1 (blue) and T_2 (red) which have been mapped into the road network. We first store them as $T_1 = \{19, 3, 5, 1, 32, 6\}$ and $T_2 = \{2, 5, 1, 4\}$. After sorting incrementally (the trajectory can be recovered using the graph connectivity information and a linked list), we can compress the list using delta encoding [73] – $T_1 = \{1, 3-1, 5-3, 6-5, 19-6, 32-19\} = \{1, 2, 2, 1, 13, 13\}$. The sorting is also fast for distance computation in Section 4.*

3.2 k-paths Trajectory Clustering

Definition 6. *(k-paths **Trajectory Clustering**) Given a set of trajectories $\{T_1, T_2, \cdots, T_n\}$, the k-paths trajectory clustering aims to partition the n trajectories into k ($k \le n$) clusters $S = \{S_1, S_2, \cdots, S_k\}$ so as 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, Dist *is the trajectory distance measure.*

The key differences between k-paths and k-means are threefold. First, trajectories can be of varying lengths instead of fixed-length vectors in a Euclidean space. Second, a trajectory distance measure *Dist* for two trajectories must be defined. Third, 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 [58], we can choose an existing trajectory 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

k-means can be extended to solve k-paths. We now describe how the well-known Lloyd's algorithm [47] can be applied to the k-paths problem in three steps as follows.

1) Initialization. Randomly choose k trajectories from the database 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 $a(i) = j$.[2]

3) Refinement. Update the centroid path of each cluster by choosing an existing trajectory which 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 challenges in the assignment and refinement steps when using the Lloyd's algorithm to solve k-paths:
1) The complexity of assignment is $\mathcal{O}(nk) \times \mathcal{O}(dis)$, where n is the number of trajectories in the dataset, k is the number of clusters and $\mathcal{O}(dis)$ is the complexity of distance computation which is quadratic when using existing distance measures.
2) The complexity of computing the mean in the refinement is constant in k-means, but it does not hold in k-paths. Even though we can extend k-medoids [58] to choose an existing trajectory as a new centroid path, we need to build a distance matrix with a complexity of $\mathcal{O}(n^2) \times \mathcal{O}(dis)$.

Based on the above complexity analysis, we can see that this is a major hurdle to scalable k-paths clustering. Thankfully, we can reduce the complexity by reformulating the way distance is computed. These key changes are detailed in the ensuing sections.

4. A NOVEL DISTANCE MEASURE-EBD

4.1 Edge-based Distance Measure

To eliminate the burden of specifying range threshold parameter and reduce the complexity of EDR [11] due to the ordering constraint, we propose a novel distance measure-EBD for trajectories on a road network. Since the order in point-based trajectory has been integrated into every edge in the directed road network G after map matching, e.g., two opposite point-pairs (p_1, p_2) and (p_2, p_1) will correspond to two different edges in G, we define EBD based on edge intersection to maintain the ordering constraint as below:

$$\text{EBD}(T_1, T_2) = \max(|T_1|, |T_2|) - |T_1 \cap T_2| \qquad (2)$$

where $|T_1|$ is the number of edges in T_1, and can also be used as the travel distance of entire trajectory in order to distinguish long edges from short ones (We discuss this in Appendix B). $T_1 \cap T_2$ denotes the intersected edges of two trajectories. Moreover, EBD obeys the non-negativity, identity of indiscernibles and symmetry.[3]

EXAMPLE 2. *As shown in Figure 1, $T_1 = \{19, 3, 5, 1, 32, 6\}$ and $T_2 = \{2, 5, 1, 4\}$. Then $|T_1| = 6$, $|T_2| = 4$ and they have two intersected edges $\{1, 5\}$, the EBD distance between T_1 and is: EBD$(T_1, T_2) = \max(6, 4) - 2 = 4$. Table 3 shows the distance computation of EDR between T_1 and T_2 based on dynamic programming. Here we apply the penalty policy of EDR by adding 1 to the distance when the starting vertices of two edges are not the same, instead of when two points' distance is bigger than the range threshold [11]. EDR returns the same distance with EBD but in a more complex way.*

Table 3: EDR distance computation between T_1 and T_2.

		19	**3**	**5**	**1**	**32**	**6**
	0	1	2	3	4	5	6
2	1	↘1	→2	3	4	5	5
5	2	2	3	↘2	3	4	5
1	3	3	3	3	↘2	3	4
4	4	4	4	4	3	↘3	→**4**

4.2 Intersection with Sorted Lists

Computing the list intersection for two integer arrays has a complexity of $\mathcal{O}(n^2)$ when these two sequences are unsorted, while two sorted lists will further reduce the complexity from quadratic to quasilinear ($\mathcal{O}(n \log n)$) using iterative binary search [15]. This is an order of magnitude improvement over unsorted lists, and fast set intersection is a fundamental problem and being continuously explored [18, 13], which leaves space to improve the efficiency of EBD in the future. Hence we use $\mathcal{O}(\text{EBD})$ to denote the complexity of EBD.

Sorting the trajectory lists not only accelerates the distance computation in Equation 2, but also reduces the storage costs as the data can be delta coded [73] (see Example 1). Hence, we pre-process all the trajectories by sorting in monotonically increasing order before clustering. Note that it is not necessary to store the original trajectory, as the sorted array can be recovered easily based on the road network.

4.3 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 [24], and reduce the number of distance computations required in practice (see the detailed applications in Section 5.2). Therefore, EBD guarantees to satisfy the triangle inequality.

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

$$\begin{aligned} \textit{EBD}(T_1, T_2) + \textit{EBD}(T_2, T_3) &\geq \textit{EBD}(T_1, T_3) \\ |\textit{EBD}(T_1, T_2) - \textit{EBD}(T_2, T_3)| &\leq \textit{EBD}(T_1, T_3) \end{aligned} \qquad (3)$$

PROOF. The proof is shown in Appendix A.1. □

Any metric index such as a VP-tree [68], M-tree [12], or MVP-tree [6] 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 unncessary distance computations

[2]In the rest of the paper, indices i and j always refer to trajectory and cluster indices, respectively.
[3]`https://en.wikipedia.org/wiki/Metric_space`

and accelerate assignments (Section 5.2). After 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 [4, 5]. We compare two different centroid initialization approaches: 1) randomly choose k trajectories from dataset; 2) k-means++ [4], and find that random initialization is sufficient for fast convergence when using EBD for k-paths clustering (Check Appendix C.2 for details). Hence, the centroid initialization of k-paths is not further explored in this work.

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 the distance for every cluster's centroid path μ_j, as shown in line 4 to 9 of Algorithm 1 in the first iteration. Then the overall complexity of each iteration will be $\mathcal{O}(nk) \times \mathcal{O}(\mathsf{EBD})$.

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 $\mu = \{\mu_1, \cdots, \mu_k\}$;
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                  $lb(i,j) \leftarrow \mathsf{EBD}(T_i, \mu_j)$;
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 $\max\big(lb(i), \frac{cb(a'(i))}{2}\big) > ub(i)$ 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                         $lb(i,j) \leftarrow \mathsf{EBD}(T_i, \mu_j)$;
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 8) and update $\mu_j$;
28     $t \leftarrow t + 1$;
29 return $\{\mu_1, \dots, \mu_k\}$;
```

Pruning by Lower Bounds. Reducing the complexity for a single distance computation can greatly 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 [22] and the previous distance computation [19, 54] is a widely adopted method in k-means.

We now show how to extend the bounding approach to 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 ($1 \le j \le k$ 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, as shown in line 7.

1) Centroid Drift. For every trajectory T_i, we maintain (1) the lower bound distance to the previous centroid paths μ_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 μ_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., $lb(i,j) = |lb(i,j) - cd(j)|$, $ub(i) = ub(i) + cd(a'(i))$, and guarantee that T_i cannot be assigned to the centroid path μ_j ($a(i) \neq j$) if $ub(i) < lb(i,j)$, denoted as:

$$lb(i,j) > ub(i) : a(i) \neq j \tag{4}$$

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 $S_{a'(i)}$ directly (line 16) if $lb(i) > ub(i)$, denoted as:

$$lb(i) > ub(i) : a(i) = a'(i) \tag{5}$$

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. Same method will apply to $ub(i)$ if $\mathsf{EBD}(T_i, \mu_{a'(i)})$ is computed.

2) Centroid Bound. For every centroid path, we compute the distance with all other $k-1$ centroid paths and build the distance matrix over the centroid paths. It can be finished immediately as k is always small. Further, we also 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 $S_{a'(i)}$:

$$\begin{gathered} \frac{\mathsf{EBD}(\mu_{a'(i)}, \mu_j)}{2} > ub(i) : a(i) \neq j \\ \frac{cb(a'(i))}{2} > ub(i) : a(i) = a'(i) \end{gathered} \tag{6}$$

To this end, we can combine two bounds to induce further pruning:

$$\max\big(lb(i), \frac{cb(a'(i))}{2}\big) > ub(i) : a(i) = a'(i) \tag{7}$$

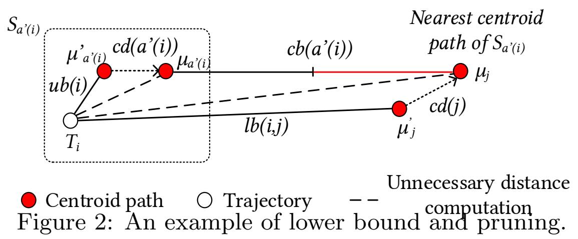


Figure 2: An example of lower bound and pruning.

Example 3. *As shown in Figure 2, trajectory T_i was assigned to $S_{a'(i)}$ in previous iteration.*[4] *Now the two new centroid paths are updated to new centroids, e.g., $\mu_j^{'} \to \mu_j$, 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 $S_{a'(i)}$. 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 [58], choosing the existing trajectory as the centroid path can make the result a real path in the road network. 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 with the following objective function:

$$O_j = \underset{\mu_j \in S_j}{\arg\min} \sum_{T \in S_j} \mathsf{EBD}(T, \mu_j) \tag{8}$$

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 $\mathcal{O}(|S_j|^2) \times \mathcal{O}(\mathsf{EBD})$ where $|S_j|$ is the number of trajectories in the cluster S_j. To reduce the complexity, we further transform the objective function as follows for EBD by Equation 2:

$$\begin{aligned} O_j &= \underset{\mu_j \in S_j}{\arg\min} \big(\sum_{T \in S_j} max(|T|, |\mu_j|) - \sum_{e \in \mu_j} \|e\| \big) \\ &= \underset{\mu_j \in S_j}{\arg\min} \big(\sum_{T \in S_j} |T| + \sum_{T \in S_j^{'}} (|\mu_j| - |T|) - \sum_{e \in \mu_j} \|e\| \big) \\ &= \underset{\mu_j \in S_j}{\arg\min} \big(\|G_j\| + \sum_{T \in S_j^{'}} (|\mu_j| - |T|) - \sum_{e \in \mu_j} \|e\| \big) \end{aligned} \tag{9}$$

where $|T|$ is the length of trajectory T, and $\|e\|$ is the frequency of the edge e, i.e., the number of trajectories crossing e in cluster S_j. $S_j^{'}$ 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 the *edge histogram* we will build. Moreover, $\|G_j\|$ is a constant and can be pre-computed by building the *length histogram.*

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

5.3.1 Histogram Construction

To compute the objective function for every trajectory in the cluster using Equation 9, 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 weight of each trajectory in Equation 12.

Length Histogram. The length histogram (LH) mainly works for computing the second part of Equation 9. For every entry, the key is the length of trajectories, and 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 8 to:

$$\begin{aligned} O_j &= \underset{\mu_j \in S_j}{\arg\min} \big(\|G_j\| + \sum_{l=1}^{|\mu_j|} (|\mu_j| - l) LH_j[l] - \sum_{e \in \mu_j} EH_j(e) \big) \\ &= \underset{\mu_j \in S_j}{\arg\min} \big(\|G_j\| + ALH_j[|\mu_j|] - \sum_{e \in \mu_j} EH_j(e) \big) \end{aligned} \tag{10}$$

where ALH_j is the <u>a</u>ccumulated <u>l</u>ength <u>h</u>istogram built from LH_j by:

$$ALH_j[m] = \begin{cases} 0, 1 \le m \le \min_{T \in S_j} |T| \\ ALH_j[m-1] + \sum_{1 \le l \le m} LH_j[l], \\ m \le \max_{T \in S_j} |T| \end{cases} \tag{11}$$

5.3.2 Selecting Trajectories as Centroid Paths

By Equation 10, 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 we maintain.

Note that we still need to check the aggregated distance in Equation 10 of every trajectory to choose the minimum one, so the running time will increase as the size of the dataset increases. Computing a lower bound for every trajectory T before computing the objective value by Equation 8 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:

$$\sum_{e \in T} EH_j(e) < UB_1(T) = \sum_{e \in T} EH_G(e) \tag{12}$$

$$\sum_{e \in T} EH_j(e) < UB_2(T) = \sum_{1 \le l \le |T|} EH_j[l] \tag{13}$$

These two bounds can be combined as $\max(UB_1(T), UB_2(T))$ to prune a trajectory before computing Equation 10.

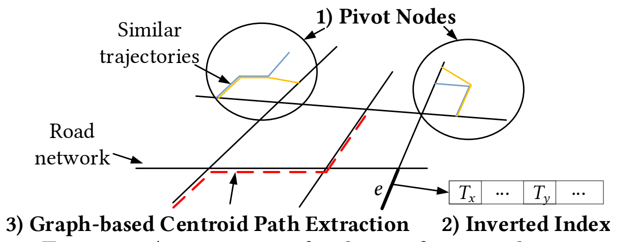


Figure 3: An overview of indexing framework PIG.

5.4 Performance Discussions

The proposed baseline can avoid distance computations to some extent by using bounding and histogram techniques during assignments (Equation 4 and 7), and refinement (Equation 12 and 13). 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 the 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 14). To avoid such cases and make the refinement more robust, choosing a complete real path of moderate length is crucial.

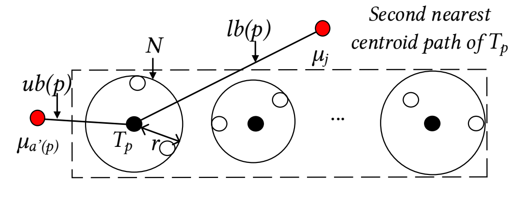
In next section, we will propose an indexing framework called PIG, which is composed of three modules: a Pivot-table, an Inverted index, and a Graph traversal. Figure 3 illustrates the basic idea of our proposed PIG, PIG further reduces distance computations using an inverted index on each edge e of the road network, and assigns pivot nodes in the Pivot-table which bounds a set of similar trajectories together instead of assigning them individually. Further, we convert the refinement to a more scalable graph traversal problem-CPEP without scanning the trajectory dataset again, and prove that it is NP-hard by reducing to the k-TSP problem [3, 53]. Finally, we propose a robust, practical and efficient greedy algorithm for the refinement.

6. BOOSTING K-PATHS WITH PIG

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 clustering.

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 equals 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.


Second nearest centroid path of T_p
$lb(p)$
N
μ_j
$ub(p)$
$\mu_{a'(p)}$
T_p
r
...
●Centroid path ●Pivot trajectory ○Trajectory
Figure 4: Batch assignment with Pivot-table.

6.2 Pivot-table for Metric Feature

In this subsection we will present an index which groups similar trajectories to accelerate the assignment process in the baseline method proposed in Section 5. Before introducing our solution, it is noteworthy that Kanungo et.al.[36] 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 [30]. 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 [9]. 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 $dist(q, T_p) - r > min_{dist}(q)$, where $min_{dist}(q)$ is the current minimum distance. Assignment can also 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 4 and 6:

$$lb(p, j) - r > ub(p) + r : a(N) \neq j \tag{14}$$

$$\frac{\mathsf{EBD}(\mu_{a'(p)}, \mu_j)}{2} - r > ub(p) + r : a(N) \neq j \tag{15}$$

Before comparing the bound with every centroid path, we use a global bound similar to the one described in Equation 7.

$$\max\left(lb(p), \frac{cb(a'(p))}{2}\right) - r > ub(p) + r : a(N) = a'(p) \tag{16}$$

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 16; otherwise, we scan the trajectories inside the node further and proceed with assignments. The centroid paths μ_j pruned by Equation 14 and 15 are no longer checked for the child trajectories of node N.

Example 4. *As shown in Figure 4, we have created a pivot-table 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.*

To group all trajectories into nodes, an M-tree [12] 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 16, 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), we use the baseline in Section 5. For a cluster with more than k trajectories, we will keep performing k-paths clustering until the number of trajectories inside is no bigger 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 refining when a node is assigned to a cluster, as we do not need to read all of the trajectories.

Note that Algorithm 2 can be also 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 unknown number of potential bus routes, where the capacity of a route is always given. We give more details on the choice of k in Appendix D.

```
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 8, we define the following problem with an objective function to find a path in a road network instead of an existing trajectory.

Definition 7. ***(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:*

$$O_j^G = \underset{\mu_j^G \in G_j}{\arg\min} \sum_{T \in S_j} EBD(T, \mu_j^G)$$
$$= \underset{\mu_j^G \in G_j}{\arg\min} \left(\|G_j\| + \text{ALH}_j[|\mu_j^G|] - \sum_{e \in \mu_j^G} \text{EH}_j(e) \right) \quad (17)$$

where $\mu_j^G \in G_j$ denotes 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, this is because each trajectory in cluster S_j is a path in G_j and used to build the frequency graph G_j of S_j, so such a path can be found in G_j to achieve same objective value in Equation 8. Moreover, a new path which is composed by the connected edges with high frequency in the graph can be scanned, and further reduce the objective value.

A straightforward method to answering CPEP is to find all the candidate paths in the graph. 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 estimated time as CPEP is NP-hard by converting it to the *k minimum Travel Salesman Problem* (k-TSP) [25, 3, 53].

Lemma 2. *The CPEP which finds a path μ_j^G in graph G_j for O_j^G is NP-hard.*

Proof. The proof is shown in Appendix A.2. □

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 which makes the whole 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. As shown in Algorithm 3, we first initialize all of the 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) in the path 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 [14]. We do not insert this kind of path into the priority queue. Moreover, a consistent path should be only expanded to a beginning and end edges, and the connectivity information these two edges (be, ee) can be retrieved from the graph G. Further, we will also check whether $EH_j.contains(e)$ as there is no trajectory crossing this edge in the cluster if not.

1) Priority Queue. Similar to Dijkstra's algorithm [17] for finding the shortest path, we use a priority queue PQ 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

[5]The k here is the number of edges and set to be $|\mu_j^G|$, it is a different concept with the k in k-paths.

current best centroid path. To achieve a tighter threshold potential, the temporally 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:

$$LB(ps) = \|G_j\| - \sum_{e \in ps} EH_j(e) + \min_{l=|ps|+1}^{l_{max}} \big(ALH_j[l] - \phi(EH_j, l)\big) \tag{18}$$

where $\phi(EH_j, l)$ is the maximum possible edge weight for the appending path with a length $l_{max} - l$, we can let $\phi(EH_j, l) = \sum_{z=1}^{l_{max}-l} EH_j[z]$. The heaviest edge is always chosen, which in turn decrease the overall bound.

3) Repeatability. Moreover, to avoid repetitive scanning of paths which share the same beginning and end edges, we build a buffer B 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 beginning 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 as shown in Equation 18, the whole algorithm will terminate and return the best path (line 7), as all the unscanned paths' best cases are impossible to beat the current best path. However, the gap between the best and threshold potential can be hard to determine according our experiments, hence we terminate if the best path does not change after a fixed number of iterations it_{max} (5,000 is sufficient according to our experiments), and the experiment shows that we achieve a better objective value than Equation 8 within thousands of iterations, i.e., it can find a better path than the dataset scanning based refinement method.

7. EXPERIMENTS

Experimental Goals. We will verify three aspects below:
1) EBD's efficiency and effectiveness compared with the widely used distance measure EDR [11] which is the most related one to ours in term of calculation;
2) EBD based k-paths and streaming k-paths's superior efficiency over state-of-the-art [8];
3) k-paths with EBD's effectiveness compared with existing distance measures and state-of-the-art [8, 43].

Datasets. We use three real-world trajectory datasets to evaluate k-paths. Table 4 shows the statistics of these three datasets, Figure 5 shows their underlying road networks[6].
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 [70]: trips for 30,000 taxis in Beijing over 3 months;
3) N-camera: a one-day (01/07/2018) camera dataset in a city, each vehicle has one trajectory. GPS-based dataset

[6]https://www.openstreetmap.org
[7]http://www.geolink.pt/ecmlpkdd2015-challenge/whoweare.html

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 $e \in \mathrm{EH}_j$ do
3      if $min > LB(e)$ then
4          $PQ.push(e, LB(e))$;
5  while $PQ$.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 $\mathrm{EH}_j.contains(e)$ and $e \notin ps$ then
11             $ps \leftarrow ps + e$;
12             Compute $O_j^G(ps)$ by Equation 17;
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 18;
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 $\mu_j$;
```

(a) Porto (b) T-drive (c) N-camera
Figure 5: Road network overview of three collections.

Porto and T-drive are mapped into road network using *GraphHopper*[8].

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 Java Virtual Machine Heap size was set to 16GB. We use *HashMap* to store the trajectories and build the inverted index, and *Google Guava MultiSet*[9] which is more efficient than HashMap for frequency counting to build the histograms.

[8]https://github.com/graphhopper/map-matching
[9]https://github.com/google/guava

Table 4: Summary of datasets and road networks.

	Porto	T-drive	N-Camera
#trajectories	1,565,595	250,997	680,550
#total edges	100,995,114	59,360,981	8,654,661
#edges per trajectory	65	237	13
average travel length (m)	5,632	31,056	6,825
Space (MB) of raw D	1,853	752	141
#Edges	150,761	126,827	1,900
#Vertices	114,099	54,198	440
Average edge length (m)	119	217	525
Space (MB) of G	11	5	0.2

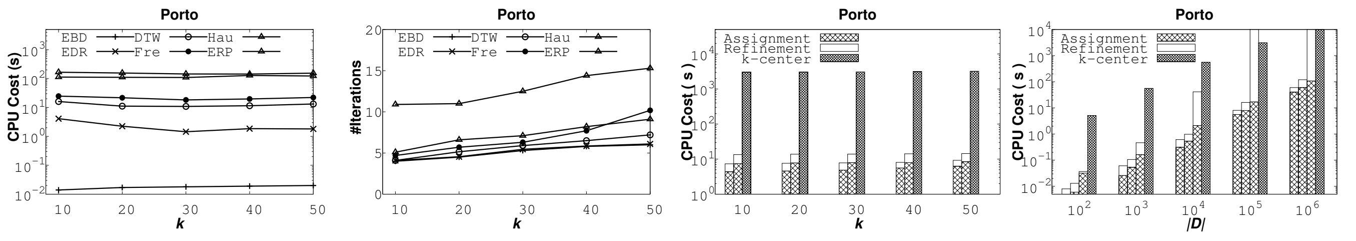


Figure 6: The single iteration clustering time and #iterations of 6 measures-based k-paths, EBD-based k-paths.

Table 5: Time (μs) for pair-wise distance computations.

	EBD	EDR	DTW	Fré	Hau	ERP
Porto	0.88	20.2	107.3	180.9	125.1	172.5
T-drive	2.56	165.4	1,940	3,026	1,991	2,948
N-camera	0.08	0.6	3.5	5.2	4.0	6.9

Table 6: Precision of similarity search (SS), assignment (Ass), and objective value (Obj) based on EDR.

	Porto			T-drive		
	SS	Ass	Obj	SS	Ass	Obj
$k = 10$	0.786	0.950	0.972	0.625	0.972	0.992
$k = 20$	0.860	0.892	0.955	0.852	0.919	0.989
$k = 30$	0.880	0.898	0.948	0.878	0.925	0.991
$k = 40$	0.893	0.869	0.936	0.898	0.869	0.985
$k = 50$	0.902	0.891	0.918	0.912	0.860	0.976

Comparisons & Measures. We mainly compare with the mostly cited trajectory clustering work TRACLUS [43] which is density-based, and the state-of-the-art k-center for point-based trajectories [8] which is partition-based. We also choose five existing trajectory distance measures widely used in partition-based clustering as shown in Table 1, and integrate them with k-paths. Specifically, DTW [67], Discret Fréchet Distance (Fré for short) [1], and EDR [66], Hausdorff (Hau for short) [8], ERP [60] are compared on k-paths with EBD. Running time, precision, convergence (objective value and running time in each iteration) and pruning power (#pruned distance computations) are reported.

7.1 Evaluation of EBD

Efficiency. Six distance measures are compared in Table 5. We build the distance matrix of three datasets by setting $|D| = 1,000$ as other five measures are too slow to produce results when setting $|D| = 10,000$, and record the time on pair-wise distance computations. We compute EDR using the method described in Table 3, and use the starting vertex of each edge in a trajectory to compose the point-based trajectory for DTW, Hau, Fré and ERP computations, which also works for our comparison k-center [8] in the next section.

Effectiveness. As shown in Table 6, we evaluate EBD's effectiveness by comparing with EDR in terms of the objective value, assignment in one iteration of k-paths and similarity search. Specifically, *SS* is computed based on the percentage of top-k searched results from EDR [64], *Ass* is computed based on the average percentage of assigned trajectories in each cluster from EDR by choosing the same centroid paths, *Obj* is computed based on the objective value of Equation 1.

Observation 1. *Table 5 shows that EBD is the fastest among all six distance measures. EBD achieves a two orders of magnitude improvement over the other five measures, especially for T-drive which stores long trajectories. For this collection, EBD only needs $\frac{1}{80}$ of EDR's time, and almost $\frac{1}{1000}$ of the time on other distance measures. Moreover, Table 6 shows that a trajectory can be assigned to a same centroid path with 90% precision when using EBD compared with EDR.*

This observation enables us to further use EBD to support efficient and accurate k-paths clustering.

7.2 Efficiency Study of k-paths

For objective comparisons, we run experiments on the same dataset with the same randomly selected initial centroid paths in each test for the efficiency and effectiveness validations. We have randomly generated the seed pool which contains 200 groups of initial centroid paths, and each group contains 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 other distance measures based k-paths for all three datasets. We set $|D| = 1,000$ based on the 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 EDR uses our inverted index approach from Section 6.1. All other distance measures employ basic assignments and refinements based on a distance matrix. We also show the number of iterations of all 6 measures. The result is presented in Figure 6.

k-paths with EBD. The histograms in Figure 6 shows the assignment and refinement time respectively, and compares 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 [8]: which is the state-of-the-art for point-based trajectory clustering, which adopts the Hausdorff distance measure [56] 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 $k = \{10, 20, \underline{30}, 40, 50\}$ and data scale $|D| = \{100, 1000, 10{,}000, \underline{100{,}000}, 1{,}000{,}000\}$. The underlined number is the default value when testing multiple parameters.

Breakdown of The Assignment Process. Figure 7 shows the number of distance calculations in the assignment process using our proposed solutions as they are grad-

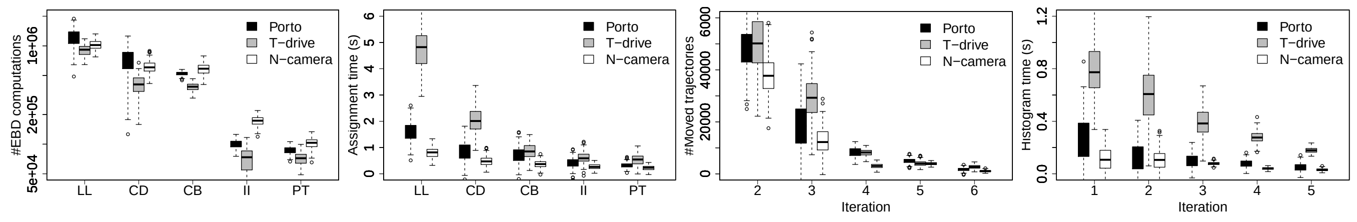


Figure 7: Assignment Breakdown: #EBD computation and time, #moved trajectories and histogram update time.

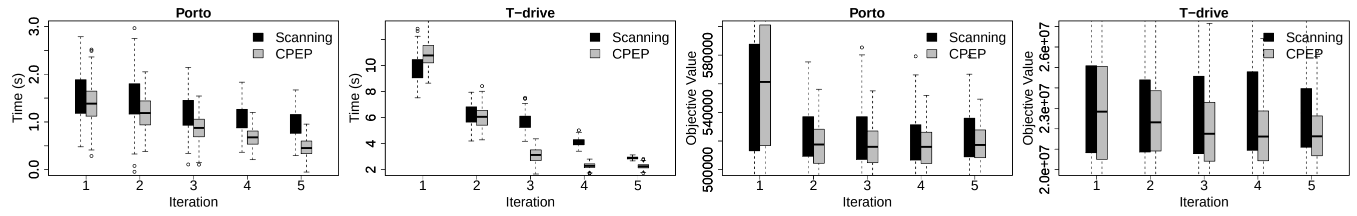


Figure 8: Refinement Breakdown: time and objective value in each iteration for scanning and CPEP.

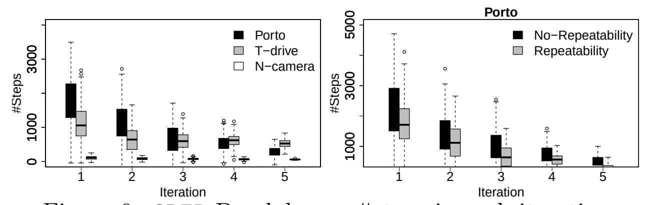


Figure 9: CPEP Breakdown: #steps in each iteration.

ually 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 The Refinement Process. Figure 8 compares the refinement time and objective value changes in each iteration. Here "Scanning" represents the 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 17 than Equation 8's), and the running time of refinement is also smaller than the full scan based method. Moreover, Figure 9 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.

Index Construction. Table 7 shows the time spent on index and histogram construction. To compress the dataset and index, we use an open source library *JavaFastPFOR*[10] to compress the sorted lists as shown in Figure 1.

Observation 2. *Firstly, EBD is the only distance measure which can achieve million-scale k-paths trajectory clustering for all six measures, while the other five measures can only achieve thousand-scale clustering. Secondly, k-paths with EBD and our proposed pruning ideas outperform the state-of-the-art method–k-center [8] 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. Thirdly, 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. Finally, with our network-based modeling, the compression techniques help reduce the total trajectory dataset size from 1.8GB to 178MB for Porto, and from 752MB to 82MB for T-drive.*

[10]`https://github.com/lemire/JavaFastPFOR`

Table 7: 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	0.5	-	0.08	0.4	-

7.3 Effectiveness Study of k-paths

Cluster Validation. 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 [61] (unsupervised). Since there are still no labeled city-wide vehicle trajectories for classification experiments ([51] 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* [29, 46]. These methods indicate how closely related objects are in a cluster, and how distinct or well separated a cluster is from other clusters.

We use multiple measures used for point clustering validity such as *Dunn* [20], *Silhouette* [59], *DavidBouldin* [16], *CalinskiHarabasz* [7] evaluated by Liu et al. [46]. The other three basic indices which measure compactness are also used. The definition for each measure can be found in Appendix E. In Table 8, we compare with the state-of-the-art distance

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

	Porto							T-drive						
CVI	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

measures DTW, EDR, ERP, Fréchet distance, and Hausdorff distance. Since the distance value varies, we normalize the results by the maximum distance for all of the distance measures. k-center [8] is also used as a baseline.

Case Study by Visualization. As shown in Figure 10, we cluster the taxi datasets to help plan new bus routes as described in the introduction. We compare five different methods, four of them are partition-based (the output will be k paths with a different color), and the fifth is the density-based method TRACLUS [43]. Since the original TRACLUS algorithm cannot be used to cluster million-scale datasets, we optimize TRACLUS using our data modeling method (See Appendix F). More visualization-based case-studies for the T-drive dataset can be found in Appendix C.1.

Observation 3. *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. An EBD-based k-paths with CPEP method can show the trend clearly, especially for T-drive dataset, while the density based-TRACLUS algorithm [43] 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. Interestingly, Figure 10 and 14 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 (EDR and k-center) do not. This is consistent with our common sense that taxis are one of the main modes of transportation around the airport, which also validates that our clustering result can show real demands, and help guide a transportation department in designing new routes.*

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 novel 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.

9. ADDITIONAL AUTHORS

10. REFERENCES

[1] 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.

[2] 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.

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

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

[5] B. Bahmani, R. Kumar, and S. Vassilvitskii. Scalable K-Means++. *PVLDB*, 5(7):622–633, 2012.

[6] T. Bozkaya and M. Ozsoyoglu. Indexing large metric spaces for similarity search queries. *ACM Transactions on Database Systems*, 24(3):361–404, 1999.

[7] T. Calinski and J. Harabasz. A dendrite method for cluster analysis. *Communications in Statistics*, 3(1):1–27, 1974.

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

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

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

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

[12] 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.

[13] R. Cohen, L. Katzir, and A. Yehezkel. A minimal variance estimator for the cardinality of big data set intersection. In *KDD*, pages 95–103, 2017.

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

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

[16] D. L. Davies and D. W. Bouldin. A cluster separation measure. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1(2):224–227, 1979.

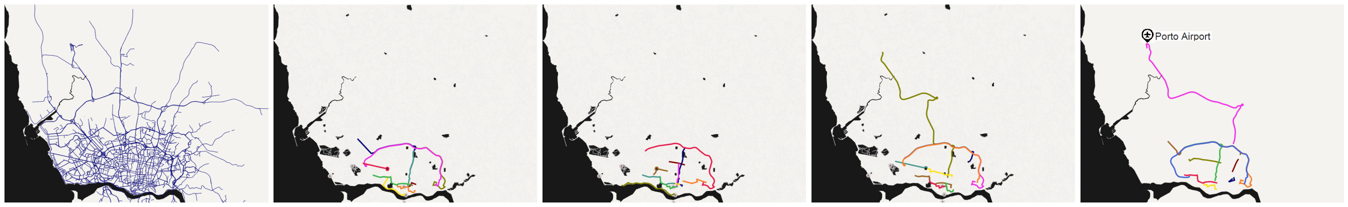


Figure 10: Porto: TRACLUS, EDR-based k-paths, k-center, EBD-based k-paths, EBD-based k-paths with CPEP.

[17] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959.

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

[19] 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.

[20] J. C. Dunn. Well-separated clusters and optimal fuzzy partitions. *Journal of Cybernetics*, 4(1):95–104, 1974.

[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] 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.

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

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

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

[27] 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.

[28] D. Habich, M. Hahmann, and W. Lehner. Parameterfreies Clustering mittels Multi-Level Ansatz. In *LWA*, pages 36–43, 2007.

[29] M. Halkidi, Y. Batistakis, and M. Vazirgiannis. On clustering validation techniques. *Journal of Intelligent Information Systems*, 17(2-3):107–145, 2001.

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

[31] 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.

[32] 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.

[33] 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.

[34] Z. G. Ives. Technical perspective - k-Shape : Efficient and accurate clustering of time series. *SIGMOD Record*, 45(1):68, 2016.

[35] 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.

[36] 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.

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

[38] E. Keogh, S. Lonardi, and C. A. Ratanamahatana. Towards parameter-free data mining. In *KDD*, pages 206–215, 2004.

[39] 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.

[40] A. Kharrat, I. Popa, K. Zeitouni, and S. Faiz. Clustering algorithm for network constraint trajectories. In *SDH*, pages 631–647, 2008.

[41] 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.

[42] B. Krogh and C. S. Jensen. Efficient in-memory indexing of network-constrained trajectories. In *SIGSPATIAL*, pages 17:1–17:10, 2016.

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

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

[45] 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.

[46] 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.

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

[48] 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.

[49] 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.

[50] 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.

[51] B. Morris and M. Trivedi. Learning trajectory patterns by clustering: Experimental studies and comparative evaluation. In *CVPR*, pages 312–319, 2009.

[52] M. D. Morse and J. M. Patel. An efficient and accurate method for evaluating time series similarity. In *SIGMOD*, pages 569–580, 2007.

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

[54] J. Newling and F. Fleuret. Fast k-means with accurate bounds. In *ICML*, pages 936–944, 2016.

[55] P. Newson and J. Krumm. Hidden Markov map matching through noise and sparseness. In *SIGSPATIAL*, pages 336–343, 2009.

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

[57] J. Paparrizos and L. Gravano. k-Shape: Efficient and accurate clustering of time series. In *SIGMOD*, pages 1855–1870, 2015.

[58] 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.

[59] 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.

[60] 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.

[61] 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.

[62] 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.

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

[64] 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.

[65] 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.

[66] 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.

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

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

[69] 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.

[70] 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.

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

[72] Y. Zheng. Trajectory data mining: An overview. *ACM Trans. Intell. Syst. Technol.*, 6(3):1–41, 2015.

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

APPENDIX

Appendices

A. PROOFS

A.1 Proof of Lemma 1

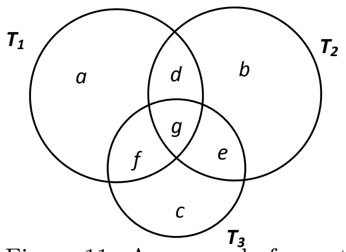


Figure 11: An example for metric proof, where each circle denotes the length, such as $|T_1| = a + d + g + f$.

Proof. Figure 11 shows the intersections a, b, c, d, e, f, g among three trajectories, all of which have a value no less than 0. $\mathsf{EBD}(T_1, T_2) = max(a + f, b + e)$, $\mathsf{EBD}(T_2, T_3) = max(b + d, c + f)$, $\mathsf{EBD}(T_1, T_3) = max(a + d, c + e)$. We will prove $\mathsf{EBD}(T_1, T_2) + \mathsf{EBD}(T_2, T_3) \geq \mathsf{EBD}(T_1, T_3)$, $max(a + f, b + e) + max(b + d, c + f) \geq max(a + d, c + e)$, and $|\mathsf{EBD}(T_1, T_2) - \mathsf{EBD}(T_2, T_3)| \leq \mathsf{EBD}(T_1, T_3)$, $|max(a + f, b + e) - max(b + d, c + f)| \leq max(a + d, c + e)$. 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 $\mathsf{EBD}(T_1, T_2)+\mathsf{EBD}(T_2, T_3) \geq \mathsf{EBD}(T_1, T_3)$ 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, $|\mathsf{EBD}(T_1,T_2)-\mathsf{EBD}(T_2,T_3)| \leq \mathsf{EBD}(T_1,T_3)$ is proved. $\square$

A.2 Proof of Lemma 2

PROOF. We can convert Equation 17 as follows:

$$\begin{aligned} O_j^G &= \underset{\mu_j^G \in G_j}{\arg\min} \left(\|G_j\| + ALH_j[|\mu_j^G|] - \max \sum_{e\in\mu_j^G} EH_j(e)\right) \\ &= \underset{|\mu_j^G|}{\arg\min} \left(\|G_j\| + ALH_j[|\mu_j^G|] - kTSP(|\mu_j^G|, G_j)\right) \end{aligned} \tag{19}$$

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) [25, 3, 53] (*Given a weighted graph, find a path of minimum*[11] *weight that passes through any k vertices*). 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 [25, 3], 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*.

B. LENGTH-SENSITIVE EBD

In a road network, the length varies from different edges, Figure 12 shows the histograms of all the edge length of Porto and T-drive. Then, the length of each edge can be integrated into EBD as:

$$\mathsf{EBD}(T_1,T_2) = \max(|T_1|,|T_2|) - |T_1 \cap T_2| \tag{20}$$

where $|T| = \sum_{e\in T} \ell(e)$, $\ell(e)$ is the length of the edge e.

With a length on every edge, EBD is still metric by using the same approach in Appendix A.1 for the proof. In the assignment, since EBD is metric, then we do not need to change the bound. In the refinement, all of the edges will be attached with the length before computing the weight with the histograms, the complexity will not increase as the length information is small and can be maintained in memory.

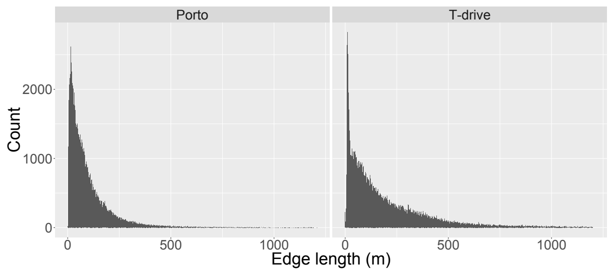


Figure 12: Edge length histograms of two datasets.

C. MORE EXPERIMENTAL STUDIES

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

C.1 Experiments on T-drive

Figure 13 shows an additional efficiency study for k-paths on T-drive, which has similar trends to the Porto dataset. As shown in Figure 14, TRACLUS on the T-drive dataset returns discrete road segments, which are hard to connect as a real path. Moreover, 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 messy 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.

C.2 Centroid Initialization

Figure 15 shows comparisons with k-means++ [4] 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 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.

C.3 Scalability

Data Scale. Since we do not have access to any larger real-world dataset, to test the scalability, we double the datasets and collect the running time, as shown in Figure 16.

k-paths with Updates. Different from streaming k-paths, 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 to the size of the collection.

D. CHOICE OF K

As k is the single parameter in the problem and our processing framework, we have two methods to estimate k. First, we can find the "elbow" [39, 41], i.e., the k where the gradient of objective value starts to decrease. As shown in Figure 17, we increase k to observe the value changes. For Porto, the elbow is around $k = 21$ (where the arrow points to), and the elbow is quite clear for T-drive, at around $k = 13$. Moreover, we can infer the best k value for some real applications such as clustering taxi trips to design bus routes. Since a bus route always has a capacity threshold, if a cluster has a large number of trajectories greater than the threshold, we will divide this cluster into k clusters, same as the pivot-table construction described in Section 6.2.2.

E. CLUSTER VALIDITY INDICES (CVI)

In Table 9, we show the formal definition of each CVI.[12] The "↑" after each CVI name shows that the higher the value, the better the cluster quality, while "↓" corresponds

[12]The μ^* in *Calinski-Harabasz* is the centroid path of whole dataset, we run k-paths by setting $k = 1$ to get it.

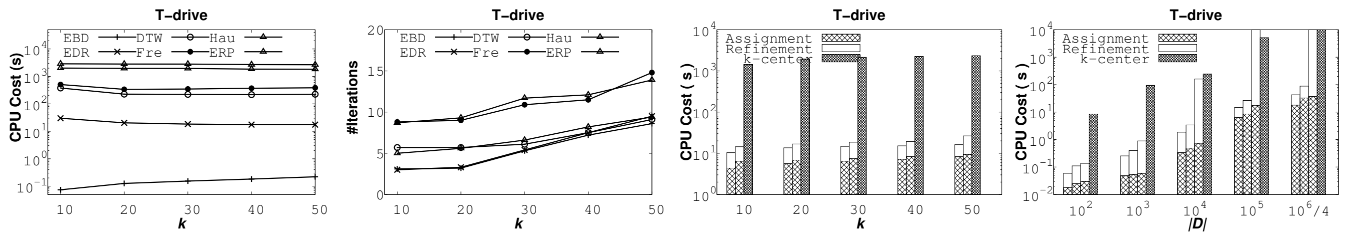


Figure 13: Single iteration clustering time, #iterations for six measure based k-paths and EBD-based k-paths.

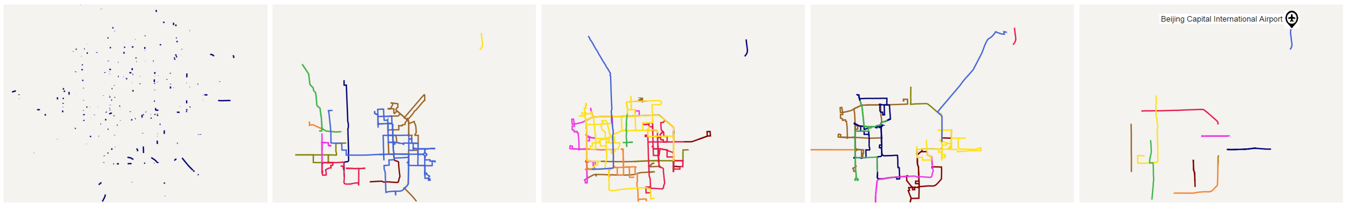


Figure 14: T-drive: TRACLUS, EDR-based k-paths, k-center, EBD-based k-paths, EBD-based k-paths with CPEP.

Table 9: 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
AverageDistance↓	$\frac{1}{\sum_{S_i \in S} \lvert S_i \rvert^2} \sum_{S_i \in S} \sum_{T_x, T_y \in S_i} Dist(T_x, T_y)$	Compactness
SquareDistance↓	$\sqrt{\{\sum_{S_i \in S} \sum_{T \in S_i} Dist^2(T, \mu_i)\} / \{\sum_{S_i \in S}(\lvert S_i \rvert - 1)\}}$	Compactness
Silhouette↑	$\frac{1}{k} \sum_{S_i \in S}(\frac{1}{\lvert S_i \rvert} \sum_{T \in S_i} \frac{b(T)-a(T)}{\max(b(T),a(T))})$, $a(T) = \frac{1}{\lvert S_i \rvert} \sum_{T_z \in S_i} Dist(T, T_z)$, $b(T) = \min_{S_j \in S - S_i} \{\frac{1}{\lvert S_j \rvert} \sum_{T_z \in S_j} Dist(T, T_z)\}$	Compactness + Separation
Dunn↑	$\min_{S_i \in S} \min_{S_j \in S} \frac{\min_{T_x \in S_i, T_y \in S_j} Dist(T_x, T_y)}{\max \max_{T_x, T_y \in S_z} Dist(T_x, T_y)}$	Compactness + Separation
Calinski-Harabasz↑	$\frac{\sum_{S_i \in S} Dist^2(\mu_i, \mu^*)}{\sum_{S_i \in S} \sum_{T \in S_i} Dist^2(T, \mu_i)} \frac{k-1}{\lvert D \rvert - k}$	Compactness + Separation
Davies-Bouldin↓	$\frac{1}{k} \sum_{S_i, S_j \in S} \max_{j \neq i}(\frac{\frac{1}{\lvert S_i \rvert} \sum_{T \in S_i} Dist(T, \mu_i) + \frac{1}{\lvert S_j \rvert} \sum_{x \in S_j} Dist(T, \mu_j)}{Dist(\mu_i, \mu_j)})$	Compactness + Separation

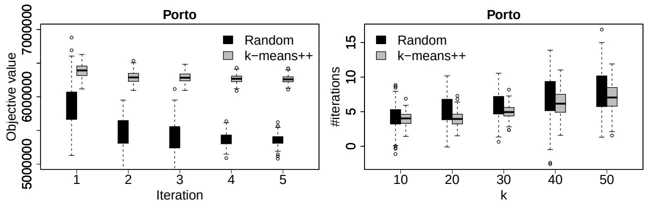


Figure 15: The effect of centroid path initialization.

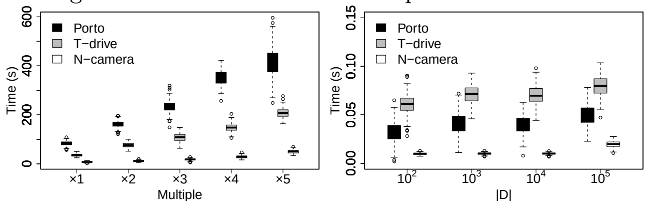


Figure 16: Scalability test on data scale and updates.

to worse cluster quality. It is noteworthy that various CVIs have certain limitations in different application scenarios [46], 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.

F. OPTIMIZED TRACLUS

TRACLUS [43] is composed of two steps: partition 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. [43] set it as the average frequency by default), which can be done with our edge histogram.

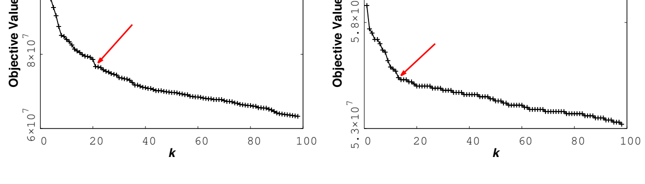


Figure 17: Finding the elbow of the objective value.