# KMCT: $k$-Means Clustering of Trajectories Efficiently in Location-Based Services

Yuanjun Liu
Soochow University
School of Computer Science and Technology
Suzhou, China
yuanjun-liu@qq.com

Guanfeng Liu
Macquarie University
School of Computing
Sydney, Australia
guanfeng.liu@mq.edu.au

Qingzhi Ma*
Soochow University
School of Computer Science and Technology
Suzhou, China
qzma@suda.edu.cn

Zhixu Li
Fudan University
School of Computer Science and Technology
Shanghai, China
zhixuli@fudan.edu.cn

Shiting Wen
NingboTech University
School of Computer and Data Engneering
Ningbo, China
wensht@nit.zju.edu.cn

Lei Zhao
Soochow University
School of Computer Science and Technology
Suzhou, China
zhaol@suda.edu.cn

An Liu
Soochow University
School of Computer Science and Technology
Suzhou, China
anliu@suda.edu.cn

## Abstract

With the widespread use of GPS devices and the advancement of location-based services, a vast amount of trajectory data has been collected and mined for various applications. Trajectory clustering, which categorizes trajectories into distinct groups, is the fundamental functionality of trajectory data mining. The challenge is how to cluster on a mass of trajectory data efficiently and universally with satisfying results. The raw trajectory clustering algorithms are universal, but trapped in the dilemma between efficiency and desirable results. Other approaches, such as density-based, road network-based, and deep learning-based algorithms, encounter issues like high time complexity, loss of trajectory integrity, reliance on road networks, and data quality during training. To tackle these challenges, we first propose the efficient KMCT ($k$-Means Clustering of Trajectories) algorithm based on a semantic interpolation transformation to cluster raw trajectories and achieve satisfying results. Additionally, we introduce the DA-KMCT (Density Accelerated $k$-Means Clustering of Trajectories) algorithm to further boost the clustering process based on trajectory densities and an optimized centroid selecting strategy. Moreover, we present a novel clustering evaluation method called IOD, which efficiently estimates clustering results on large-scale datasets with linear time complexity. Experimental results on real-world datasets demonstrate that KMCT and DA-KMCT outperform five related methods in terms of clustering quality and time efficiency, and the proposed IOD evaluation shows a strong correlation with the Silhouette Coefficient, offering a reliable and efficient alternative for evaluating clustering results.

*Corresponding author.

## CCS Concepts

• **Information systems → Clustering**; **Location based services**.

## Keywords

Trajectory Clustering, Trajectory Data Mining, Spatial Dataset

**ACM Reference Format:**
Yuanjun Liu, Guanfeng Liu, Qingzhi Ma, Zhixu Li, Shiting Wen, Lei Zhao, and An Liu. 2024. KMCT: $k$-Means Clustering of Trajectories Efficiently in Location-Based Services. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management (CIKM '24), October 21–25, 2024, Boise, ID, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3627673.3679848

## 1 Introduction

With the widespread availability of GPS devices and the increasing use of location-based services [24, 14, 4], massive volumes of trajectory data are being collected from the activities of people and vehicles. Trajectory data, comprising sequences of spatial points, are beneficial for numerous real-life applications, including urban computing [9, 31], transportation [39, 10], and recommendation [18, 42, 38]. Trajectory clustering, an essential and popular task in trajectory data analytics, plays a vital role in many real-world applications, including privacy protection [7, 34], traffic management [22, 17], anomaly detection [41, 29], and mobile pattern mining [21, 22, 13].

The challenge lies in efficiently clustering trajectory data. On the one hand, efficiency is crucial for processing vast amounts of trajectory data. For example, DiDi, the largest ride-sharing company in China, collects more than 106 TB of trajectory data daily to provide services such as route planning, travel time estimation, and urban capacity analyses [9]. The company must process these trajectories efficiently to identify subtle changes in users' preferences and update their strategies daily to serve the clients effectively. On the other hand, producing effective results is crucial for driving the aforementioned applications and providing better strategies to attract more clients.

Given $N$ trajectories, where each trajectory is a sequence of $n$ spatial points $T = [p_1, p_2, \ldots, p_n]$, trajectory clustering algorithms aim to classify these trajectories into $k$ clusters such that trajectories within each cluster are close to each other. Existing trajectory clustering algorithms typically extend general clustering algorithms by incorporating spatio-temporal attributes specific to trajectories and can be categorized based on the preprocessing steps before clustering.

Density based algorithms [20, 16] partition trajectories into segments, then utilize general density clustering algorithms like the DBSCAN algorithm [8] to obtain clusters of segments and merge these segments into complete trajectories. However, these algorithms have drawbacks. On one hand, their time complexity of $O(Nn\log(Nn))$ is expensive for handling massive trajectory data. On the other hand, they compromise the integrity of the entire trajectory and may generate false trajectories that violate real-world rules, particularly in complex scenarios such as cloverleaf interchanges.

Road network based algorithms [33, 28, 16] typically begin with map matching. While these algorithms generally have lower time complexity compared to those operating in Euclidean space, mapping trajectories to the network results in the loss of original trajectory information, and the matching process may introduce additional errors. Additionally, these algorithms necessitate road network data specific to the trajectories' locations, which may not be available in many scenarios, such as maritime trajectories.

Deep learning based algorithms [10, 19, 32, 35] approach trajectory clustering by modeling trajectories using neural networks and clustering based on the representation vectors generated by these networks. However, their performance heavily relies on the quality of the training data and requires significant time and resources for training, rendering them inefficient for large-scale datasets.

Clustering raw trajectories maintains trajectory integrity and does not require additional settings like segmentation, map matching, and training. However, existing algorithms struggle to achieve both efficiency and desirable results simultaneously. One of the simplest methods involves combining the $k$-medoids algorithm with trajectory similarity measures such as DTW [3] and ITS [25]. Nevertheless, computing the similarity of each pair demands $O(N^2)$ time, which can be computationally expensive. Additionally, the $k$-means algorithm cannot be directly applied to trajectory data because there is no mean operation defined for trajectory data. Existing algorithms address this issue by transforming trajectories into vectors using various methods to utilize the $k$-means algorithm. However, these transformations often fail to preserve sufficient spatial and semantic information, leading to unsatisfactory results.

Moreover, clustering evaluations play a crucial role in parameter selection and statistical analysis. However, existing evaluations such as the Silhouette Coefficient have a time complexity of $O(N^2)$, which can be even more time-consuming than the clustering process itself. This limitation makes them unsuitable for assessing large-scale datasets efficiently.

To efficiently cluster raw trajectories while achieving desirable results, we propose a novel trajectory clustering algorithm called KMCT (***k*-M**eans **C**lustering of **T**rajectories). This algorithm transforms trajectories into vectors using a fast interpolation strategy that preserves spatial and semantic information. It then clusters trajectories based on the $k$-means algorithm. Additionally, we introduce the DA-KMCT algorithm (**D**ensity **A**ccelerated ***k*-M**eans **C**lustering of **T**rajectories), which further accelerates the clustering process by leveraging trajectory densities and an optimized centroid selection strategy. Both clustering algorithms have a time complexity of $O(Nnkt)$.

To address the inefficiency of the Silhouette Coefficient, we propose a novel and fast evaluation metric called IOD (**I**nner-**O**uter **D**istance rate) to assess clustering results. The IOD metric computes approximate distances within and outside clusters in linear time.

In summary, our paper makes the following contributions:

- We propose two novel algorithms KMCT and DA-KMCT to cluster trajectories in $O(Nnkt)$ time, where $N$ is the number of trajectories, $n$ is the average number of points in trajectory, and $t$ is the number of iterations.
- We compare KMCT and DA-KMCT with five related methods using two real-world datasets. Experimental results demonstrate that KMCT and DA-KMCT achieve the most desirable results with minimal time consumption.
- We introduce the IOD evaluation metric with linear time complexity, providing an efficient means to estimate clustering results on large-scale datasets. Experimental analyses show a strong correlation between IOD and the Silhouette Coefficient.

The rest of the paper is organized as follows. The related work is discussed in Section 2. Section 3 presents the KMCT algorithm, and Section 4 presents the DA-KMCT algorithm. The experiments are elaborated in Section 5. Finally, we summarize our work in Section 6.

## 2 Related Work

The $k$-means algorithm [26] is widely used in general clustering tasks. It assigns data to the nearest cluster centroid and updates centroids iteratively by computing the mean of each cluster until convergence or reaching a maximum number of iterations. The time complexity of $k$-means is $O(Nhkt)$, where $t$ represents the number of iterations and $h$ denotes the dimension of data. However, applying the $k$-means algorithm directly to trajectory data faces challenges due to the unique characteristics of trajectories, such as varying lengths, diverse point positions, and disparate sampling rates, making meaningful mean value calculation difficult for trajectories, which hinders the direct use of the $k$-means algorithm for clustering trajectories.

To fully utilize the $k$-means algorithm, various methods transform trajectories into a unified representation that allows for the computation of a mean value. The HeatMap algorithm [23] discretizes space into grids and then projects each trajectory onto the grid to generate a heatmap, serving as the representation vector of the trajectory, which requires $O(Ngkt)$ time complexity, where $g$ denotes the number of grids. Similarly, the VectorField algorithm [11], after discretizing space into a grid, computes a vector field as the representation based on the positions and velocities of points, with a time complexity of $(Ngkt)$. The EMLR algorithm [36] employs linear regression to fit the trajectory curves, while the tDPMM algorithm uses the fast Fourier transform to extract trajectory features. Both convert trajectories into $h$ dimensional vectors and perform clustering in $O(Nhkt)$ time. The EigMtx algorithm [12] employs eigenvalue decomposition on the similarity matrix to construct the feature matrix, treating each row as the vector of each trajectory, thus requiring $O(N^2s)$ to build the matrix. While these methods convert trajectories into fixed-length vectors and then invoke the $k$-means algorithm, they either suffer from time-consuming conversions or struggle to capture significant features for effective clustering results. The SOM-TC algorithm [6] defines a mean operation for trajectories based on GeoHash strings, but computing on strings takes longer than working with numerical vectors.

The $k$-medoids algorithm is another well-known clustering algorithm that differs from the $k$-means algorithm by determining the datum closest to other data within each cluster as the new center, rather than using the mean value of data. This algorithm requires $O(N^2s)$ time to calculate the distances between all data pairs. Due to its reliance on distance calculations rather than mean computation, the $k$-medoids algorithm is suitable for trajectory clustering using various trajectory similarity measures such as DTW [3], ERP [5], ITS [25], etc. The time complexity $s$ for trajectory similarity measures is typically $s = n^2$, and $s = n$ for ITS [25].

The above clustering methods are based on the distance between data, while another group of methods performs clustering based on density. The DBSCAN algorithm [8] and the DPC algorithm [30] treat high-density data as centers, with density calculated based on the number of data within a given range. After finding the initial centers, the algorithms select surrounding data whose densities are higher than a predetermined threshold to join the clusters. If a tree structure [15] is used to search nearby data, these algorithms have a time complexity of $O(N \log N)$. Existing methods [20, 16, 22] divide trajectories into segments using the MDL (Minimum Description Length) criterion and define the density of segments, then employ the density-based clustering algorithms such as DBSCAN to cluster the segments, finally combine segments from the same cluster into a trajectory. These algorithms efficiently find common sub-trajectory segments. However, the time complexity of $O(Nn \log(Nn))$ is not efficient enough on massive trajectory data. Additionally, they cannot classify entire trajectories.

With the advancement of deep learning, neural networks are utilized for clustering in two ways. The first group of methods [35, 21] trains the network by other tasks to obtain the representation vectors of trajectories, such as masked point complementing and similar trajectory retrieving under noise. After training, they invoke general clustering algorithms such as the $k$-means algorithm. The second group integrates the process of computing trajectory representation with the clustering process [10, 19, 32]. They update the model parameters while clustering, and the end-to-end architecture helps leverage the feedback from the clustering stage to optimize the representations.

The aforementioned methods cluster in space, while some approaches cluster on the road network. The $k$-paths algorithm [33] first matches trajectories to the road network, then defines the similarity of road trajectories, ultimately clusters based on the $k$-means algorithm with histograms of edges and distance constraints in $O(Nntk)$ time. The NEAT algorithm [16] segments trajectories based on the road network and performs the DBSCAN algorithm. The LBTC algorithm [28] divides trajectories into segments based on the road network and uses the label propagation algorithm for clustering in $O(Nn + Vt)$ time, where $V$ represents the number of nodes in the network.

There is an amount of work to improve the clustering by seeding carefully. k-means++ [1] generates good seeds in $O(Nhk)$ time, AFK-MC$^2$ [2] approximates the k-means++ [1] algorithm in $O(Nh + k^3 h \log k)$ time, and LDDW-K-means [27] employs the local density to initialize seeds in $O(N^2h)$ time. Note that both the above algorithms and our proposed DA-KMCT algorithm can accelerate the clustering, but they focus on different points. On the one hand, the DA-KMCT algorithm improves the procedure of clustering instead of seeding. On the other hand, the total running times with seeding algorithms are even higher than the DA-KMCT algorithm, as shown in the experiments.

## 3 The KMCT Algorithm

The $k$-means algorithm is known for its efficiency and effectiveness. However, applying $k$-means directly to trajectory data is challenging due to the inability to calculate the mean of trajectories naturally. In order to apply the $k$-means algorithm to trajectory data, trajectories need to be transformed into a unified representation to calculate their mean.

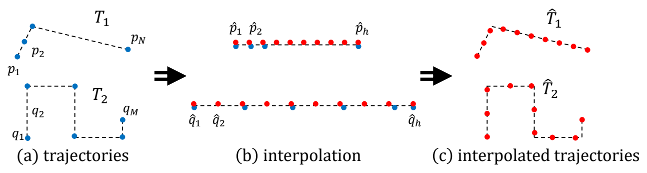

**Figure 1: The interpolation process.**

ITS [25] is a trajectory similarity measure with linear time complexity and state-of-the-art performance. It transforms trajectories of varying lengths into fixed-length interpolated trajectories through an interpolation process and calculates similarity based on Euclidean distance. Figure 1 shows the interpolation process. There are two trajectories with different numbers of points and various shapes, and the interpolated trajectories have the same number of points and retain the shapes of trajectories. Since each interpolated trajectory has $h$ points, recording $x$ and $y$ coordinates, it can be viewed as a vector of length $2h$. This allows us to define the "mean" operation for trajectories naturally:

$$\text{mean}(\mathbb{T}) = \frac{\sum_{T_i \in \mathbb{T}} v_i}{N} \quad (1)$$

where $\mathbb{T}$ is the set of trajectories, $N = |\mathbb{T}|$ is the number of trajectories, and $v_i$ =Interpolation$(T_i, h)$ is the interpolated trajectory.

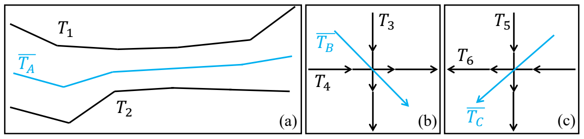

**Figure 2: The averaged trajectories maintain the spatial and semantic information. (a): $\bar{T_A}$ captures the spatial information, i.e., the position and the shape of $T_1$ and $T_2$. (b) and (c): $\bar{T_B}$ and $\bar{T_C}$ reserve the semantic information, such as the direction of trajectories.**

The interpolation process generates the trajectory in the very same spatial space and maintains the sequence and shape of trajectories, which helps the mean operation to capture the spatial and semantic information of trajectories, as Figure 2 illustrates. (a) shows the mean trajectory $\bar{T_A}$ has the averaged shape and position of $T_1$ and $T_2$, and (b) and (c) show the mean trajectories $\bar{T_B}$ and $\bar{T_C}$ reserve the sequential and semantic information of trajectories, thus their positions and directions are different.

Based on Equation (1), we propose the KMCT algorithm to apply $k$-means to trajectory data. The algorithm transforms trajectories into vectors of fixed length and then utilizes $k$-means for clustering.

Algorithm 1 shows the detail of the KMCT algorithm. Given trajectory dataset $\mathbb{T}$ and the number of clusters $k$, the algorithm returns clustering result $cls$, i.e., the classification of each trajectory. The interpolation process in the ITS algorithm requires the number of interpolated points $h$, and ITS suggests setting $h$ as the average length of all trajectories, thus line 1 calculates the average length with $O(h) = O(n)$. Then, lines 2-3 transform trajectories into vectors $\mathbf{V} = \{v_i | v_i$ =Interpolation$(T_i, h) | T_i \in \mathbb{T}\}$, and line 4 involves the $k$-means algorithm to cluster trajectories.

```
Algorithm 1: KMCT
  Input: trajectories $\mathbb{T}$, integer $k$
  Output: clustering result $cls$
1 $h \leftarrow \sum_{T_i \in \mathbb{T}} |T_i|/N$
2 for $T_i \in \mathbb{T}$ do
3     $v_i \leftarrow$Interpolation$(T_i, h)$
4 $cls \leftarrow k$-means$(\mathbf{V}, k)$
```

Algorithm 1 has $O(Nntk)$ time complexity, where $N$ is the number of trajectories, $n$ is the average number of points of all trajectories, $k$ is the number of clusters, and $t$ is the number of iterations of the $k$-means algorithm. Calculating the average number of points of trajectories in line 1 needs $O(Nn)$ time, transforming trajectories into vectors in lines 2-3 demands $O(Nn)$ time, and involving the $k$-means algorithm in line 4 requires $O(Nntk)$ time.

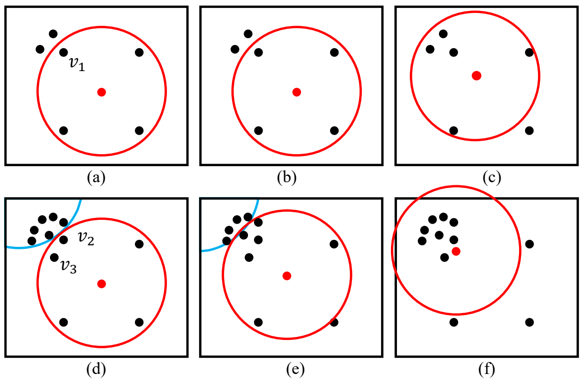

**Figure 3: The examples of accelerating clustering based on density. The black points are data points, and the red points are the clustering centroids. The data in the same red circle have the same classification. (a-c) show that the density can help to break the pitfall of saddle points. (a): The centroid is at the geometric center of the data points at the current iteration. (b): The centroid will not move if the data have no change at the next iteration, even though it's better to move to the left-top corner. (c): With the help of density, the centroid will move to the left-top corner, since the density at that area is higher. (d-f) show that the density can help the clustering centroids move more efficiently. (d): The data points and centroid of the current iteration. (e): The centroid moves slowly towards the left-top corner. (f): With the help of density, the centroid moves faster towards the optimal position.**

## 4 The DA-KMCT Algorithm

### 4.1 The Density Can Boost Clustering

The KMCT algorithm has $O(Nntk)$ time complexity, yet the real running time fluctuates with $t$, i.e., the number of iterations. To mitigate this, we introduce density-based optimizations to expedite cluster centroids convergence during clustering.

During each iteration of the $k$-means algorithm, each cluster $j$ calculates the mean of its data $\varsigma_j = \{v_i | cls_i \equiv j\}$ as the new centroid $c_j$, i.e.,

$$c_j = \frac{\sum_{v_i \in \varsigma_j} v_i}{|\varsigma_j|} \quad (2)$$

This approach treats each datum equally, but it may lead to inefficiencies as demonstrated in Figure 3.

Figure 3.(a-c) illustrates that Equation (2) may fall into the pitfall of saddle points. Figure 3.(a) shows the centroid is at the geometric center of $\varsigma_j$ at the current iteration, and Figure 3.(b) shows the centroid will not change if the $\varsigma_j$ have no change in the following iterations, even though it can move to the left-top corner to achieve higher cohesion. Figure 3.(d-f) illustrates that the clustering centroid moves slowly under the direction of Equation (2). Figure 3.(d) shows the data points and centroid of the current iteration, then the centroid moves to the left-top corner slowly to positions exhibited in Figure 3.(e) and Figure 3.(f), even though it can move to Figure 3.(f) within a single iteration.

To address these challenges, we leverage density-based centroid iteration for more efficient clustering. We calculate the density

$\rho(v_i)$ for each datum $v_i$ and utilize it to update centroids as follows:

$$c_j = \frac{\sum_{v_i \in \varsigma_j} \rho(v_i) v_i}{\sum_{v_i \in \varsigma_j} \rho(v_i)} \tag{3}$$

This weighted approach assigns higher importance to denser regions, allowing centroids to move more purposefully and improving convergence speed and clustering quality.

Based on the density, the Equation (3) can tackle the problem of the examples mentioned before. Figure 3.(c) demonstrates that the centroid will move to the left-top corner, since the density of datum $v_1$ is higher than others. Therefore, the density can help to break the pitfall of saddle points. Figure 3.(f) indicates that the centroid will move to $v_2$ and $v_3$ directly from Figure 3.(d) to Figure 3.(f), since the densities there are much higher than others. Therefore, the density can help the clustering centroid move more efficiently.

## 4.2 The Density of Trajectory Data

To apply the methodology discussed above to trajectory clustering, we need to calculate the density of trajectory data, which demands the density of trajectory points. Existing methods [8, 20, 16, 27] need $O(N^2 s)$ or $O(Nn \log(Nn))$ time to calculate the density, which are more time consuming than the clustering process. In this section, we proposed an $O(Nn)$ time method to compute the density efficiently.

In a general scenario, the density of a single point $p$ is defined as the number of points within a specified range, i.e.,

$$\rho(p) = |q \in R(p)| \tag{4}$$

where $R(\cdot)$ is a square size of $r \times r$ centered at $p$.

However, when dealing with trajectory data clustering, the density of a point should not be solely based on the number of points, as trajectories can have various shapes. As shown in Figure 4, both points $p_1$ and $p_2$ have 2 trajectories within their range, but the trajectories have different shapes. Therefore, the point-wise densities of points $p_1$ and $p_2$ are different according to Equation (4). Thus, the naive definition in Equation (4) is not suitable for clustering trajectory data.

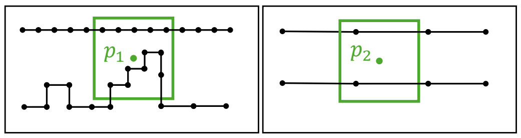

**Figure 4: Both of $p_1$ and $p_2$ have 2 trajectories in their range, but their point-wise densities differ due to the various shapes of trajectories.**

To calculate density accurately for trajectory clustering, we define the density of a point as the number of trajectories within its range, i.e.,

$$\varrho(p) = |T \in R(p)| \tag{5}$$

Given a trajectory $T$ and point $p$, if any point in $T$ is within the range of $p$, i.e., $\exists q \in T, q \in R(p)$, then trajectory $T$ is within the range of $p$, denoted as $T \in R(p)$.

To efficiently calculate point density, we use accumulated and quantized cells to compute Equation (5) in linear time. First, we quantize the space $F$ containing trajectories into cells of size $\epsilon \times \epsilon$. Second, for each trajectory, we initialize a new group of cells with

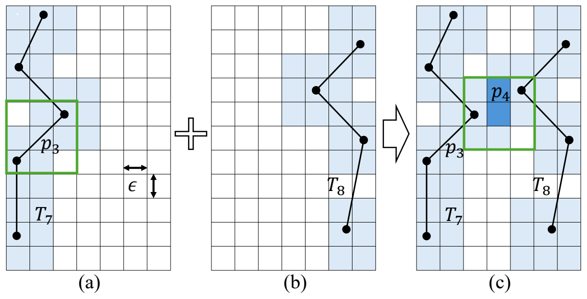

**Figure 5: Efficient density calculation using quantized cells. Two trajectories $T_7$ and $T_8$ are shown. (a,b): Trajectories quantized into groups of cells of size $\epsilon \times \epsilon$. Cells within the trajectory range have a weight of 1, while others have a weight of 0. (c): Accumulation of cell weights for density calculation. For example, point $p$ has a density of 2 since it lies within the range of two trajectories, contributing twice to its accumulated weight.**

weight 0 and set the weight of cells within the trajectory's range to 1. Finally, we accumulate the cells from all trajectories and retrieve the weight of the cell as the point's density.

Given a cell and a trajectory $T$, if the center point of the cell $q$ is in the range of any point in $T$, i.e., $\exists p \in T, q \in R(p)$, then the cell is in the range of $T$.

Figure 5 illustrates an example of computing the densities of two points $p_3$ and $p_4$. Figure 5.(a) and Figure 5.(b) show two groups of cells of two trajectories respectively, and Figure 5.(c) shows the accumulated cells. Note that there are two points in the range of $p_3$, but the weight of the cell of $p_3$ is 1, which avoids the problem of repetitive computation displayed in Figure 4.

Based on the density of points, we define the density $\rho_i$ of trajectory $T_i$ as the average density of points in $T$, i.e.,

$$\rho_i = \rho(T_i) = \frac{\sum_{p \in T_i} \varrho(p)}{|T_i|} \tag{6}$$

Figure 6 provides an example of computing the densities of trajectories $T_9$, $T_{10}$, and $T_{11}$. After quantization and accumulation, the cells and their weights are displayed. The densities of points in $T_{10}$ are 2, 2, 1, 2, and 2 respectively, thus the density of $T_{10}$ is 1.8. Similarly, the densities of $T_9$ and $T_{11}$ are both 1.4.

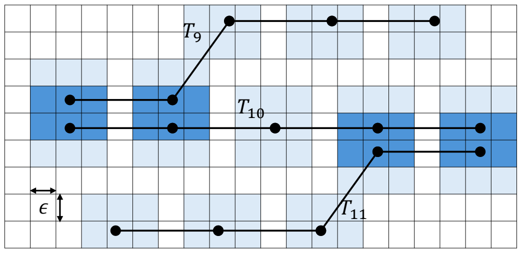

**Figure 6: Densities of trajectories $T_9$, $T_{10}$, and $T_{11}$ are 1.4, 1.8, and 1.4 respectively.**

Algorithm 2 formulates the details. Given the set of trajectories $\mathbb{T}$, the rectangle space $F$ containing trajectories, and the cell size $\epsilon$, the algorithm returns the density $\rho_i$ of each trajectory $T_i$. The algorithm

**Algorithm 2:** Densities of trajectories

```
   Input: trajectories $\mathbb{T}$, rectangle $F$, number $\epsilon$
   Output: the density $\rho_i$ of each trajectory $T_i$
 1 $Q[F]^\epsilon \leftarrow 0, M[F]^\epsilon \leftarrow 0$
 2 for $T \in \mathbb{T}$ do
 3     $M[F]^\epsilon \leftarrow 0$
 4     for $p \in T$ do
 5         $M[F \cap R(p)]^\epsilon \leftarrow 1$
 6     $Q \leftarrow Q + M$
 7 for $T_i \in \mathbb{T}$ do
 8     $\rho_i \leftarrow 0$
 9     for $p \in T_i$ do
10         $\rho_i \leftarrow \rho_i + Q[p]^\epsilon$
11     $\rho_i \leftarrow \rho_i / |T_i|$
```

first computes the densities of cells $\boldsymbol{Q}$, then computes the density of each trajectory based on $\boldsymbol{Q}$. Line 1 quantizes the space into 2 groups of cells $\boldsymbol{Q}$ and $\boldsymbol{M}$ of size $\epsilon \times \epsilon$, and initialize their weights as 0. Following operations recycle $\boldsymbol{M}$ to calculate the weights of cells of each trajectory then accumulate cells to $\boldsymbol{Q}$, instead of generating a new group of cells for each trajectory. For each trajectory $T$, line 3 initializes the $\boldsymbol{M}$ with 0, lines 4-5 find cells in the range of $T$ and set their weight as 1, then line 6 accumulates the weight of $\boldsymbol{M}$ to $\boldsymbol{Q}$. Next, for each trajectory $T_i$, lines 8-11 compute its density $\rho_i$ based on Equation (6). Specifically, line 10 gets the density of each point $p$ by retrieving the weight of the cell that $p$ lies, and line 11 computes the average density of all points.

Algorithm 2 is $O(Nn)$ time complexity, where $N$ is the number of trajectories, and $n$ is the number of points of each trajectory. Lines 2-6 consume $O(Nn)$ time to get $\boldsymbol{Q}$, and lines 7-11 spend $O(Nn)$ time to compute the $\rho_i$ of each trajectory $T_i$.

Note that cells are well-structured, allowing lines 5 and 10 in the Algorithm 2 to be executed parallelly. Thus, our algorithm can be accelerated by SIMD techniques [1] easily, to process multiple cells simultaneously. There are many practical tools that can optimize the algorithm automatically, such as the modern GCC compiler [2] and the NumPy library [3].

## 4.3 Efficient Clustering on Trajectory Data Based on Density

Based on Equation (6), we can optimize clustering centroids by incorporating trajectory densities into Equation (3), i.e.,

$$c_j = \frac{\sum_{v_i \in \varsigma_j} \rho_i v_i}{\sum_{v_i \in \varsigma_j} \rho_i} \tag{7}$$

where $v_i$ is the representation vector, as well as the interpolated trajectory of trajectory $T_i$, $\rho_i = \rho(T_i)$ is the density of trajectory $T_i$, $\varsigma_j$ is the set of trajectories in the cluster $j$, and $c_j$ is the centroid of cluster $j$.

[1]https://en.wikipedia.org/wiki/Single_instruction,_multiple_data
[2]https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html
[3]https://numpy.org/doc/stable/reference/simd/index.html

**Algorithm 3:** DA-KMCT

```
  Input: trajectories $\mathbb{T}$, integer $k$
  Output: clustering result $cls$
1 invoke Algorithm 2 to calculate the density $\rho_i$ of each
    trajectory $T_i \in \mathbb{T}$
2 invoke lines 1-3 of Algorithm 1 to transform each trajectory
    $T_i$ into vector $v_i$
3 $\boldsymbol{c} \leftarrow$the initialized $k$ centers
4 while $\boldsymbol{c}$ (almost) do not change do
5     for $T_i \in \mathbb{T}$ do
6         $cls_i \leftarrow \arg\min_{1 \le j \le k} d(v_i, c_j)$
7     for $j \in [1, 2, \dots, k]$ do
8         update $c_j$ according to Equation (7)
```

Algorithm 3 shows the detail of the DA-KMCT algorithm. Given the set of trajectories $\mathbb{T}$, the number of clusters $k$, the rectangle space $F$ containing trajectories, and the cell size $\epsilon$, the algorithm returns the classification $cls$ of each trajectory. Firstly, the algorithm invokes Algorithm 2 to compute the density $\rho_i$ of each trajectory $T_i$, and involves Algorithm 1 to transform trajectories into vectors. Then, the algorithm enhances the $k$-means algorithm based on Equation (7). Line 3 initializes $k$ centers $\boldsymbol{c} = \{c_j | j \in [1, 2, \dots, k]\}$, and subsequent operations continue untill $\boldsymbol{c}$ converges. Lines 5-6 find the nearest centroid $j$ as the classification of trajectory $T_j$, then lines 7-8 update each centroid $c_j$ according to Equation (7).

Algorithm 3 has a time complexity of $O(Nntk)$. Lines 1-2 require $O(Nn)$ time, and the enhanced $k$-means algorithm in lines 3-8 takes $O(Nntk)$ time. The time complexity of the DA-KMCT algorithm is the same as the KMCT algorithm, but the $t$ of the DA-KMCT algorithm will be smaller thanks to Equation (7). However, computing density requires additional space for cells, since the space complexity of Algorithm 2 is $\frac{F}{\epsilon^2}$. Therefore, the KMCT algorithm is suitable for general scenarios, and the DA-KMCT algorithm is preferable on computers with more than $\frac{F}{\epsilon^2}$ space.

## 5 Experiment

We evaluate our algorithms with five baselines on two real-world datasets. Section 5.1 provides details about the datasets, parameters, evaluations, and the compared methods. Section 5.2 displays the time consumption and the efficiency of compared methods. Moreover, we analyze the proposed evaluation in Section 5.2.2.

## 5.1 Experimental Setting

*5.1.1 Datasets and Parameters.* Table 1 shows two real-world trajectory datasets collected in Beijing, China. T-drive[37] contains 1e4 trajectories from 1e4 taxis from Feb. 2 to Feb. 8, 2008, and GeoLife[40] contains 1.7e4 trajectories from 182 users from Apr. 2007 to Aug. 2012.

**Table 1: Dataset statistics**

| Dataset | #Trajectories |
|---|---:|
| GeoLife[40] | 17,621 |
| T-drive[37] | 10,357 |

**Table 2: Parameters**

| Parameter | Values | Default |
|---|---|---|
| $N$ | [1,000, 2,000, 3,000, 4,000, 5,000] | all |
| $k$ | [10, 20, 40, 80, 160] | $\log N$ |
| $r$ | [50, 100, 150, 200, 250] meter | 100 |
| $\epsilon$ | [10, 20, 30, 40, 50] meter | 10 |

The parameter settings are displayed in Table 2. The number of trajectories $N$ varies from 1,000 to 5,000 for the main experiment, and is set to all for analyses. The number of clusters is set as $\log N$ for the main experiment, and varies from 10 to 160 for analyses. The DA-KMCT algorithm requires the range size $r$, the cell size $\epsilon$, and the space $F$. The $r$ varies from 50 to 250 meters and is set as 100 meters by default. The $\epsilon$ varies from 10 to 50 meters and is set as 10 meters by default. The $F$ is set as the main urban areas, i.e., the $5^{th}$ Ring Road of Beijing within longitude $\in [116.25°, 116.5°]$ and latitude $\in [39.8°, 40.05°]$, about $25{,}000^2$ square meters.

*5.1.2 Evaluations.* The SC (Silhouette Coefficient) is widely used in the unsupervised clustering of small-scale datasets. The silhouette score of each datum $v_i$ is defined as:

$$s(v_i) = \begin{cases} \dfrac{b(v_i) - a(v_i)}{\max(a(v_i), b(v_i))}, & |\varsigma_j| > 1 \\ 0, & |\varsigma_j| \le 1 \end{cases}, \tag{8}$$

where $\varsigma_j$ is the cluster of $v_i$, $a(v_i) = \frac{1}{|\varsigma_j|-1}\sum_{v_x \in \varsigma_j} d(v_i, v_x)$ is the average distance inside the cluster and stands for the cohesion of the cluster, and $b(v_i) = \min\limits_{\varsigma' \neq \varsigma_j} \frac{1}{|\varsigma'|}\sum_{v_y \in \varsigma'} d(v_i, v_y)$ is the minimum average distance outside the cluster and stands for the dispersion with other clusters. Then, the SC for the entire dataset is calculated as $\max\limits_{\varsigma} \bar{s}(\varsigma)$, where $\bar{s}(\varsigma)$ is the averaged $s(v), v \in \varsigma$.

The SC ranges from $[-1, 1]$, with 1 indicating the best clustering result, characterized by low distance inside the cluster and high distance outside the cluster. The time complexity of the SC is $O(N^2)$, which can even be higher than the clustering algorithm itself, making it unsuitable for large amounts of.

$$s(v_i) = \begin{cases} 1 - \dfrac{a(v_i)}{b(v_i)}, & a(v_i) < b(v_i) \\ 0, & a(v_i) = b(v_i) \\ \dfrac{b(v_i)}{a(v_i)} - 1, & a(v_i) > b(v_i) \end{cases} \tag{9}$$

The SC computes the rate between average inner distance and average outer distance, as Equation (9) shows. Thus we compute the rate between approximate total inner distance and approximate total outer distance to measure the quality of clusters effectively termed IOD (Inner-Outer Distance rate), as Equation (10) shows.

$$IOD = \frac{\sum_i d(v_i, v_x)}{\sum_i d(v_i, v_y)} \tag{10}$$

where $v_x$ is any datum whose classification is as the same as $v_i$, and $v_y$ is any datum whose classification differs with $v_i$.

The IOD varies from $[0, \infty)$ with different meanings. IOD $\approx 0$ means a good result, IOD $< 1$ means a practicable result, IOD $\approx 1$ means that the result is as bad as random, and IOD $> 1$ means the result is worse than random. Therefore, a better result has a bigger SC and a smaller IOD. The IOD measures the approximate total distance to save time by choosing one pair of samples, which may be inaccurate. However, the following experiment shows that IOD is stable and as accurate as SC with large amounts of data.

**Algorithm 4:** IOD

```
Input: dataset $V$, integer $k$, clustering result $cls$
Output: IOD
1 calculate $\varsigma_j$ of each cluster $j$ based on $cls$
2 $a \leftarrow 0, b \leftarrow 0$
3 for $v \in V$ do
4     choose $v_x \in \varsigma_j, v_y \notin \varsigma_j$ uniformly
5     $a \leftarrow a + d(v_i, v_x)$
6     $b \leftarrow b + d(v_i, v_y)$
7 $IOD \leftarrow a/b$
```

The IOD is $O(N)$ time complexity as shown in Algorithm 4, making it suitable for large-scale datasets. We use both SC and IOD in the main experiment, and only IOD in analytical experiments after demonstrating the strong correlation between them.

*5.1.3 Compared Methods.* We compare our algorithms with the following five methods. KMD+ITS clusters based on the $k$-medoids algorithm with the ITS[25] similarity measure, it is $O(N^2n)$ complexity. EgiMtx[12] employs the $k$-means algorithm on the similarity matrix after eigenvalue decomposition, it is $O(N^2n)$ complexity. HeapMap[23] employs the $k$-means algorithm after transforming trajectories into heap maps, it is $O(Ngkt)$ complexity, where $g$ is the number of grids. EMLR[36] employs the EM algorithm after transforming trajectories into vectors based on linear regression, it is $O(Nhkt)$ complexity, where $h$ is the length of vectors. VectorField[11] employs the $k$-means algorithm after transforming trajectories into vector fields, it is $O(Ngkt)$ complexity.

Furthermore, we compare the DA-KMCT algorithm with SOTA seeding algorithm AFK-MC$^2$ [2] enhanced KMCT to verify both the effect and efficiency. The AFK-MC$^2$ [2] consumes $O(Nh + k^3 h \log k)$ time to find good seeds.

We implement the above algorithms in Python. We do not include the deep learning based methods, since their training times even exceed the clustering times. The following experiments demonstrate that our algorithms perform better with less time.

## 5.2 Experiment Results

*5.2.1 Main Results.* Figure 7.(a)-(b) shows the running time of all clustering algorithms. The proposed KMCT and DA-KMCT algorithms exhibit the smallest time consumption, indicating superior efficiency compared to others. The running times of all methods increase with the $N$, and the running times of $O(N^2n)$ methods are generally higher than others, but the EMLR algorithm performs worse than $O(N^2n)$ methods such as KMD+ITS on GeoLife dataset, indicating the poor quality of the transformed trajectory vectors.

Figure 7.(c)-(d) shows the SC values of all algorithms. The proposed KMCT and DA-KMCT algorithms with the highest SC scores dominate other methods on two datasets, even the $O(N^2n)$ methods, i.e., the KMD+ITS and EigMtx algorithms. The KMD+ITS algorithm is as good as the DA-KMCT algorithm but worse than the KMCT

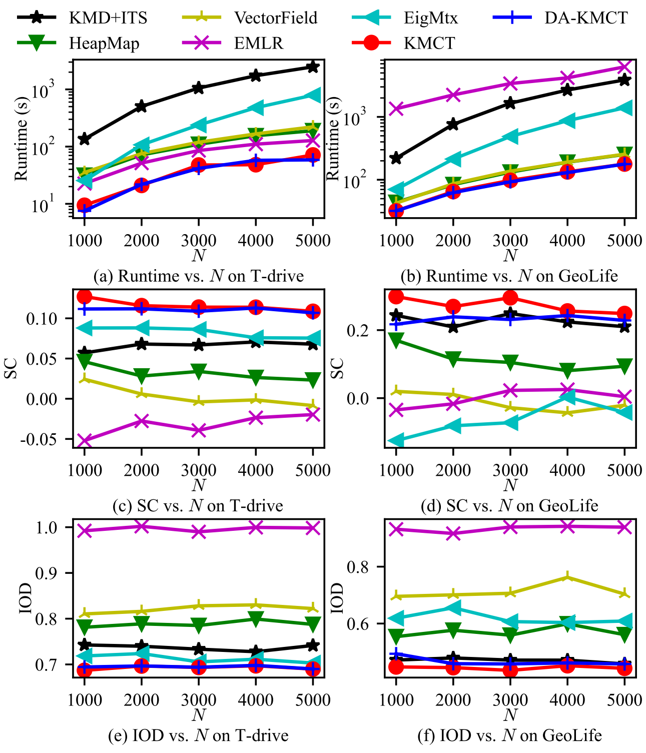

**Figure 7: Main results**

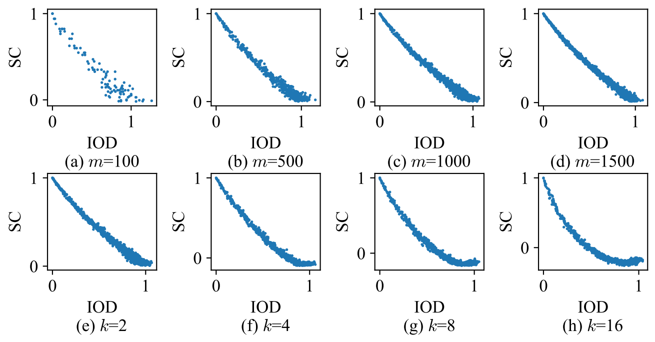

**Figure 8: IOD vs. SC**

algorithm on the GeoLife dataset, and is worse than both KMCT and DA-KMCT algorithms on the T-drive dataset. The EigMtx algorithm is better than the KMD+ITS algorithm on the T-drive dataset, but worse than others on the GeoLife dataset, indicating the instability of such an algorithm.

Figure 7.(e)-(f) shows the IOD values of all algorithms. On the one hand, the proposed KMCT and DA-KMCT algorithms have the best IOD values, which exhibit superior performance compared to others. On the other hand, compared to Figure 7.(c)-(d), the orders of algorithms of SC and IOD are similar, indicating the effectivity of the IOD metric. Specifically, the orders of algorithms on the T-drive dataset of SC and IOD both are [KMCT, DA-KMCT, EigMtx, KMD+ITS, HeapMap, VectorField, EMLR], and the orders on the GeoLife dataset only differ in VectorField and EMLR.

In conclusion, compared to other algorithms, the KMCT and DA-KMCT algorithms exhibit the best time consumption and clustering results. Notably, the DA-KMCT algorithm requires less time than the KMCT algorithm, albeit with higher IOD and lower SC values. Further analysis will be conducted in the following experiment.

*5.2.2 Analyses of IOD and SC.* The comparison between Figure 7.(c)-(d) and (e)-(f) has discovered the connection between the IOD and the SC. To further investigate the relationship between them, we conduct experiments on synthetic toy datasets in this section.

First, we generate $k$ root points in $[0, 1]$ uniformly and duplicate them until the total number of points reaches $m$. The optimal clustering result would be that all points duplicated from the same root point are in the same cluster, and the SC is 1 and the IOD is 0 in this case. Then, we do not conduct cluster algorithms, but generate the clustering results based on the optimal result, and compute the IOD and SC values on the generated results. We change the ratio of correct labels in generated clustering results gradually and record the IOD and SC values.

Figure 8.(a)-(d) illustrates the relationship between IOD and SC when $k = 2$ and $m$ ranges from 100 to 1500. When $m$ is small, i.e., $m = 100$, the relationship appears blurry as shown in Figure 8.(a), while the figure becomes clearer as $m$ increases, as depicted in Figure 8.(b)-(d). IOD and SC exhibit a strong negative correlation; specifically, high IOD corresponds to low SC, and vice versa.

Figure 8.(e)-(f) presents the relationship between IOD and SC when $m = 1,000$ and $k$ ranges from 2 to 16. When $k$ is small, i.e., $k = 2$, there is a linear relationship between IOD and SC, as illustrated in Figure 8.(e). Interestingly, when $k$ is larger, i.e., $k = 16$, SC struggles to differentiate performance when it is close to 0, whereas IOD can more effectively identify these results and provide more consistent evaluations.

*5.2.3 Analyses of the Convergence Speed.* This section exhibits the efficiency the DA-KMCT algorithm and investigates the reason. Specifically, we compare it with KMCT and SOTA seeding algorithm enhanced KMCT+AFK-MC$^2$, and the following results demonstrate the advantage of the proposed density accelerating strategy.

Figure 9.(a)-(b) displays the running time of the algorithms. The running times increase with $k$. First, the DA-KMCT algorithm surpasses the KMCT algorithm in speed, indicating that Equation (7) effectively enhances clustering. Second, the DA-KMCT algorithm is faster than the seeding algorithms enhanced KMCT, showing the superior efficiency of the DA-KMCT algorithm. Third, the KMCT+AFK-MC$^2$ is not as efficient as expected, indicating that the general improvements for the k-means algorithm are unsuitable for trajectory data.

Figure 9.(c)-(d) showcases the IOD values of the algorithms. The IOD values decrease with $k$. While the IOD values of the DA-KMCT algorithm are slightly higher than others, it achieves better efficiency by sacrificing a tiny amount of effectiveness.

Figure 3 demonstrates that density can boost clustering by analyzing the movement of singular cluster centroid theoretically, and Figure 9.(a)-(b) shows that the DA-KMCT algorithm is more efficient than the KMCT algorithm from a global view. To further investigate the effect of density, we analyze the clustering procedure to interpret the efficiency of the DA-KMCT algorithm.

We record the movement of centroids of each iteration and employ the CD (Convergence Distance) to analyze the convergence speed during the clustering procedure. Given the $j$-th centroid at

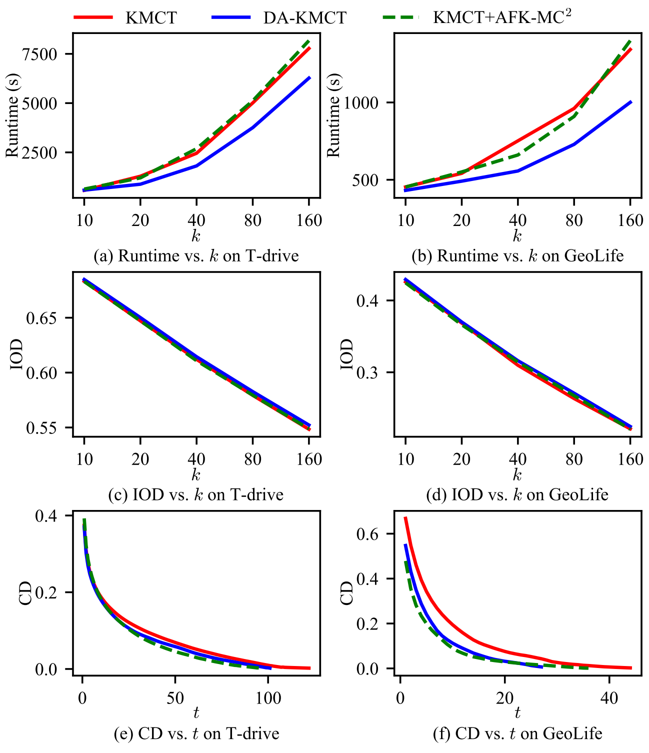

**Figure 9: Analyses of the convergence speed**

iteration $t$ of position $c_j^t$, the CD is the distance towards its final position $c_j^\infty$, as Equation (11) shows,

$$CD_j^t = d(c_j^t, c_j^\infty) \tag{11}$$

and the total CD value at iteration $t$ is calculated as the average of all centroids, i.e., $CD^t = \sum CD_j/k, 1 \le j \le k$.

Since all compared methods in this section are based on the KMCT algorithm with $O(Nntk)$, and when $N, n, k$ are fixed, a faster algorithm must have a smaller $t$, i.e., the number of iterations until convergence. Comparing the KMCT and the DA-KMCT algorithms, there are two observations when we set the same initial centroids. First, Figure 9.(e)-(f) shows the algorithms converge when its CD value reaches 0, and the $t$ of the KMCT algorithm is smaller than the DA-KMCT algorithm, which corresponds with the theoretical analysis. Second, the CD value of the KMCT algorithm descends faster than the DA-KMCT algorithms, which explains why the KMCT algorithm has a smaller $t$.

Comparing the DA-KMCT and the seeding algorithm enhanced KMCT+AFK-MC$^2$, the KMCT has a similar or smaller $t$, which demonstrates the advantage of the proposed density accelerating strategy. Comparing the KMCT and the KMCT+AFK-MC$^2$, the KMCT+AFK-MC$^2$ has a smaller $t$ but has a longer running time, which is blamed on the time of the AFK-MC$^2$ algorithm.

*5.2.4 Analyses of Parameters of the DA-KMCT Algorithm.* Figure 10.(a)-(b) illustrates the effect of the cell size $\epsilon$ on the DA-KMCT algorithm. The IOD values increase with $\epsilon$ since smaller $\epsilon$ values result in more cells, leading to more accurate densities and better results.

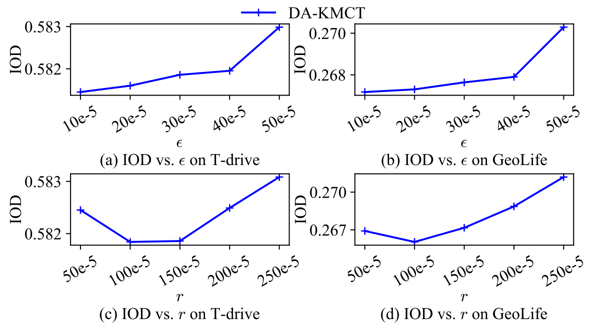

**Figure 10: The effect of parameter on the DA-KMCT algorithm**

Figure 10.(c)-(d) demonstrates the effect of the range size $r$ on the DA-KMCT algorithm. Inaccuracies in density arise with overly small or large $r$ values, underscoring the importance of choosing an appropriate $r$ for the DA-KMCT algorithm. Overall, the fluctuations caused by parameters are much less significant than the disparities between other algorithms, thus the parameters can be chosen casually with reasonable values.

## 6 Conclusion

This paper addressed the challenge of clustering a vast amount of trajectory data efficiently and effectively. Existing trajectory clustering algorithms often struggle to balance universality, efficiency, and satisfactory results. We proposed the novel KMCT algorithm, which combines the $k$-means algorithm with interpolation transformation while preserving spatial and semantic information. Additionally, we introduced the DA-KMCT algorithm to further enhance clustering by accelerating centroid selection based on trajectory densities. Furthermore, our novel clustering evaluation method, IOD, provides an efficient way to estimate clustering results with linear time complexity by approximating the inner and outer distances of clusters. Experimental results compared with five related methods on real-world datasets demonstrated that KMCT and DA-KMCT achieve satisfactory results with minimal time consumption, and IOD emerges as a viable option for evaluating large-scale datasets.

## Acknowledgments

This work is supported by Natural Science Foundation of Jiangsu Province (Grant Nos. BK20211307), by Zhejiang Provincial Natural Science Foundation of China under Grant No. LY24F020021, by Priority Academic Program Development of Jiangsu Higher Education Institutions, and by the Ningbo Science and Technology Special Projects under Grant No. 2022Z235 and 2024Z263 .

## References

[1] David Arthur and Sergei Vassilvitskii. 2007. K-means++: the advantages of careful seeding. In *SODA*, 1027–1035.
[2] Olivier Bachem, Mario Lucic, S. Hamed Hassani, and Andreas Krause. 2016. Fast and provably good seedings for k-means. In *NIPS*, 55–63.
[3] Donald J. Berndt and James Clifford. 1994. Using dynamic time warping to find patterns in time series. In *KDD*, 359–370.
[4] Bing-Jyue Chen, Chiok Yew Ho, and De-Nian Yang. 2024. After: adaptive friend discovery for temporal-spatial and social-aware xr. In *ICDE*, 2639–2652.
[5] Lei Chen and Raymond T. Ng. 2004. On the marriage of lp-norms and edit distance. In *VLDB*, 792–803.

[6] Pranita Dewan, Raghu Ganti, and Mudhakar Srivatsa. 2017. Som-tc: self-organizing map for hierarchical trajectory clustering. In *ICDCS*, 1042–1052.

[7] Yulan Dong and Dechang Pi. 2018. Novel privacy-preserving algorithm based on frequent path for trajectory data publishing. *Knowledge-Based Systems*, 148, 55–65.

[8] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. 1996. A density-based algorithm for discovering clusters in large spatial databases with noise. In *KDD*, 226–231.

[9] Ziquan Fang, Lu Chen, Yunjun Gao, Lu Pan, and Christian S. Jensen. 2021. Dragoon: a hybrid and efficient big trajectory management system for offline and online analytics. *VLDB J.*, 30, 2, (Mar. 2021), 287–310.

[10] Ziquan Fang, Yuntao Du, Lu Chen, Yujia Hu, Yunjun Gao, and Gang Chen. 2021. E2dtc: an end to end deep trajectory clustering framework via self-training. In *ICDE*, 696–707.

[11] Nivan Ferreira, James T. Klosowski, Carlos E. Scheidegger, and Cláudio T. Silva. 2013. Vector field k-means: clustering trajectories by fitting multiple vector fields. In *EuroVis*, 201–210.

[12] Zhouyu Fu, Weiming Hu, and Tieniu Tan. 2005. Similarity based vehicle trajectory clustering and anomaly detection. In *ICIP*. Vol. 2, II–602.

[13] Yunjun Gao, Ziquan Fang, Jiachen Xu, Shenghao Gong, Chunhui Shen, and Lu Chen. 2024. An efficient and distributed framework for real-time trajectory stream clustering. *TKDE*, 36, 5, 1857–1873.

[14] Yang Guo, Zhiqi Wang, Jin Xue, and Zili Shao. 2024. A spatio-temporal series data model with efficient indexing and layout for cloud-based trajectory data management. In *ICDE*, 1171–1184.

[15] Antonin Guttman. 1984. R-trees: a dynamic index structure for spatial searching. In *SIGMOD*, 47–57.

[16] Binh Han, Ling Liu, and Edward Omiecinski. 2015. Road-network aware trajectory clustering: integrating locality, flow, and density. *IEEE Trans. Mob. Comput.*, 14, 2, 416–429.

[17] Ping Han, Wenqing Wang, Qingyan Shi, and Jucai Yue. 2021. A combined online-learning model with k-means clustering and gru neural networks for trajectory prediction. *Ad Hoc Networks*, 117, 102476.

[18] Nan Jiang, Haitao Yuan, Jianing Si, Minxiao Chen, and Shangguang Wang. 2024. Towards effective next poi prediction: spatial and semantic augmentation with remote sensing data. In *ICDE*, 5061–5074.

[19] Sun Jianhua, Li Yuxuan, Fang Hao-Shu, and Lu Cewu. 2021. Three steps to multimodal trajectory prediction: modality clustering, classification and synthesis. In *ICCV*, 13230–13239.

[20] Jae-Gil Lee, Jiawei Han, and Kyu-Young Whang. 2007. Trajectory clustering: a partition-and-group framework. In *SIGMOD*, 593–604.

[21] Xiucheng Li, Kaiqi Zhao, Gao Cong, Christian S. Jensen, and Wei Wei. 2018. Deep representation learning for trajectory similarity computation. In *ICDE*, 617–628.

[22] Anqi Liang, Bin Yao, Bo Wang, Yinpei Liu, Zhida Chen, Jiong Xie, and Feifei Li. 2024. Sub-trajectory clustering with deep reinforcement learning. *VLDB J.*, 33, 3, (May 2024), 685–702.

[23] Weiyao Lin, Hang Chu, Jianxin Wu, Bin Sheng, and Zhenzhong Chen. 2013. A heat-map-based algorithm for recognizing group activities in videos. *IEEE Trans. Circuits Syst. Video Technol.*, 23, 11, 1980–1992.

[24] Yu Liu, Qian Ge, Wei Luo, Qiang Huang, Lei Zou, Haixu Wang, Xin Li, and Chang Liu. 2024. Graphmm: graph-based vehicular map matching by leveraging trajectory and road correlations. *TKDE*, 36, 1, 184–198.

[25] Yuanjun Liu, An Liu, Guanfeng Liu, Zhixu Li, and Lei Zhao. 2023. Towards effective trajectory similarity measure in linear time. In *DASFAA*, 283–299.

[26] S. Lloyd. 1982. Least squares quantization in pcm. *IEEE T. Inform. Theory*, 28, 129–137.

[27] Zhuoheng Lv and Qingxin Liu. 2023. Distance-weighted k-means clustering algorithm based on local density. In *ICAICE*, 90–96.

[28] Xinzheng Niu, Ting Chen, Chase Q. Wu, Jiajun Niu, and Yuran Li. 2020. Label-based trajectory clustering in complex road networks. *IEEE Trans. Intell. Transp. Syst.*, 21, 10, 4098–4110.

[29] Bardh Prenkaj and Paola Velardi. 2024. Unsupervised detection of behavioural drifts with dynamic clustering and trajectory analysis. *TKDE*, 36, 5, 2257–2270.

[30] Alex Rodriguez and Alessandro Laio. 2014. Clustering by fast search and find of density peaks. *Science*, 344, 6191, 1492–1496.

[31] Sijie Ruan, Xi Fu, Cheng Long, Zi Xiong, Jie Bao, Ruiyuan Li, Yiheng Chen, Shengnan Wu, and Yu Zheng. 2023. Filling delivery time automatically based on couriers' trajectories. *TKDE*, 35, 2, 1528–1540.

[32] Junjun Si, Yang Xiang, Jin Yang, Li Li, Bo Tu, Xiangqun Chen, and Rongqing Zhang. 2023. A dual self-supervised deep trajectory clustering method. In *ISCC*, 1169–1172.

[33] Sheng Wang, Zhifeng Bao, J. Shane Culpepper, Timos Sellis, and Xiaolin Qin. 2019. Fast large-scale trajectory clustering. *Proc. VLDB Endow.*, 13, 1, (Sept. 2019), 29–42.

[34] Shuo Wang, Surya Nepal, Richard O. Sinnott, and Carsten Rudolph. 2019. P-STM: privacy-protected social tie mining of individual trajectories. In *ICWS*, 1–10.

[35] Wei Wang, Feng Xia, Hansong Nie, Zhikui Chen, Zhiguo Gong, Xiangjie Kong, and Wei Wei. 2021. Vehicle trajectory clustering based on dynamic representation learning of internet of vehicles. *T-ITS*, 22, 6, 3567–3576.

[36] Jishang Wei, Hongfeng Yu, Jacqueline H. Chen, and Kwan-Liu Ma. 2011. Parallel clustering for visualizing large scientific line data. In *LDAV*, 47–55.

[37] Jing Yuan, Yu Zheng, Chengyang Zhang, Wenlei Xie, Xing Xie, Guangzhong Sun, and Yan Huang. 2010. T-drive: driving directions based on taxi trajectories. In *ACM-GIS*, 99–108.

[38] Kaiqi Zhang, Hong Gao, Xixian Han, Jian Chen, and Jianzhong Li. 2022. Maximizing range sum in trajectory data. In *ICDE*, 755–766.

[39] Kai Zheng, Bolong Zheng, Jiajie Xu, Guanfeng Liu, An Liu, and Zhixu Li. 2017. Popularity-aware spatial keyword search on activity trajectories. *WWW*, 20, 4, (July 2017), 749–773.

[40] Yu Zheng, Xing Xie, and Wei-Ying Ma. 2010. Geolife: A collaborative social networking service among user, location and trajectory. *IEEE Data Eng. Bull.*, 33, 2, 32–39.

[41] Lv Zhongjian, Xu Jiajie, Zhao Pengpeng, Liu Guanfeng, Zhao Lei, and Xiaofang Zhou. 2017. Outlier trajectory detection: a trajectory analytics based approach. In *DASFAA*, 231–246.

[42] Zhuang Zhuang, Wei Tianxin, Liu Lingbo, Qi Heng, Shen Yanming, and Yin Baocai. 2024. Tau: trajectory data augmentation with uncertainty for next poi recommendation. In *AAAI* number 20. Vol. 38, 22565–22573.