Computing (2022) 104:1949–1975
https://doi.org/10.1007/s00607-022-01075-4

REGULAR PAPER

Osprey: a heterogeneous search framework for spatial-temporal similarity

Hao Dai[1] · Yang Wang[1] · Chengzhong Xu[2]

Received: 23 October 2021 / Accepted: 24 February 2022 / Published online: 4 April 2022

Abstract

In this paper, a heterogeneous spatial-temporal similarity search framework is proposed, in which the datasets come from multiple different asynchronous data sources. Due to measuring error, data loss, and other factors, the similarity search based on single points along a trajectory usually cannot fulfill the accuracy requirements in our heterogeneous case. To address this issue, we introduce a concept of the spatial-temporal cluster of points, instead of single points, which can be identified for each target query. By following this concept, we further design a spectral clustering algorithm to construct the clusters in the pre-processing phase effectively. And the query processing is improved for the accuracy of the search by unifying multiple search metrics. To validate our idea, we also prototype a clustered online spatial-temporal similarity search system, "*Osprey*", to calculate in parallel the similarity of spatial-temporal sequences in the heterogeneous search on a distributed database. Our empirical study is conducted based on an open dataset, called "T-Drive", and a billion-scale dataset consisting of WiFi positioning records gathered from the urban metro system in Shenzhen, China. The experimental results show that the latency of our proposed system is less than 4s in most cases, and the accuracy is more than 70% when the similarity exceeds 0.5.

Keywords Spatial-temporal trajectory · Heterogeneous similarity search · Spectral clustering · Query processing

✉ Yang Wang
yang.wang1@siat.ac.cn

Hao Dai
hao.dai@siat.ac.cn

Chengzhong Xu
czxu@um.edu.mo

[1] Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences, Shenzhen, Guangdong, China

[2] Faculty of Science and Technology, University of Macau, Macau, China

Mathematics Subject Classification 68P10 (Searching and sorting) · 68P20 (Information storage and retrieval of data)

1 Introduction

With the rapid developments of sensing technologies and the explosive availability of mobile devices, massive volumes of spatial-temporal traces of moving objects, i.e., *trajectories*, are being generated and gathered for diverse digital services at an unprecedented speed [1]. In particular, given a trajectory Q, the *trajectory similarity search* refers to the query search that can find all subsets of trajectories from a certain dataset, whose *similarity factors* to Q exceed a pre-defined threshold θ [2, 3]. The trajectory similarity search is one of the fundamental operations in many space-time related applications as the found trajectories often contain rich information regarding the queried moving object, which can be exploited for various purposes [4, 5]. For example, policemen can use the trajectory of a target vehicle to detect what other moving objects once followed its tracks during a specified period of time in the criminal investigation.

Given the importance of trajectory similarity search, there exist many studies with an attempt to be more accurately approximate to real trajectories in various contexts [4–8]. However, these studies often target the *homogeneous* case where the information used in the query and the collected trajectory sequences stored in the spatial-temporal database are in general from the same data domain, or in particular, from the same data source, say both are extracted from the GPS location data. As such, they are not always effective in certain cases where the query information is heterogeneous in that they are from different data domains or data sources, which is often manifested in short of some queried data segments with respect to the stored trajectory information. A typical example is in the field of transportation, we may hope to find the most similar GPS trajectory and detailed information of a vehicle by leveraging some observed track points of the vehicle from Closed Circuit Television (CCTV) [8]. Such a kind of requirements can also be found in other fields, such as wild animals tracking [9], visitor plan recommendation [10], etc., where similar trajectories and corresponding information based on a given observation or sampling trajectory are always desired.

In this paper, we focus on the *heterogeneous spatial-temporal similarity search* problem, which is characterized by the feature that query data and database data might not come from the same single data source. Rather, they could be from multiple different asynchronous data sources in support of the query. For example, the spatial-temporal database may maintain the trajectory information extracted from GPS datasets while allowing the query to be constructed from WiFi connection data. This could happen in a case that a policeman wants to confirm if a suspect is the person who once walked alone the same path to a place during a period of time. The database has recorded the trajectories based on the person's GPS-equipped device. However, unfortunately at this time the suspect's GPS information was not available, and only his WiFi connections along a path were detected, each being maintained for a while to show his stationary point. With the heterogeneous similarity search, the policeman can retrieve the required information by comparing the trajectories from

different data sources. As with this case, the heterogeneous similarity search could be also useful in other applications and meanwhile substantially improving the quality of spatial-temporal services.

One of the main challenges from such heterogeneous search may lie in the fact that the asymmetric information between query and database makes the traditional similarity search based on point-wise fashion as in the homogeneous cases ineffective to fulfill the accuracy requirements in our settings. To address this challenge, we first deliberately extend a spectral cluster algorithm in an off-line pre-processing phase to group the spatial-temporal points in the queried area and allow a so-called *cluster*, instead of a single point, to be identified for each point in the query to search the sub-trajectory with high similarity. After the sub-trajectory in each cluster has been found, the full similar trajectory is constructed by concatenating them as a whole. To improve the accuracy of the search we also unify multiple search metrics with an attempt to enhance the robustness and stableness of the metrics.

Unfortunately, due to the involved massive computational workloads and the online requirements of the query processing, the proposed approaches may not be efficient. To cope with this problem and validate our idea, we also prototype an online spatial-temporal similarity search system, called "*Osprey*", which, by integrating the proposed algorithms in two major components–a data pre-processing component and a query processing component, calculates in parallel the similarity of spatial-temporal sequences in the search based on a distributed database.

Our empirical study is conducted based on billions of WiFi positioning records and WiFi connection data items gathered from the urban metro system in Shenzhen, China [11]. The similarity search of these two kinds of datasets can be used to not only improve the indoor positioning and but also find companions, which is very useful in security controls as we exemplified above. The experimental results show that our proposed approach can achieve relatively good performance in heterogeneous spatial-temporal similarity search in terms of both latency and accuracy. To the best of our knowledge, we are the first to conduct such a similarity search based on the WiFi positioning records and the WiFi connection data items from the urban metro system. To summarize, we make the following contributions in this paper:

- We extend a spectral clustering algorithm for heterogeneous similarity search problem where the query data and the database data might be from multiple different asynchronous data sources.
- We propose multiple metrics for heterogeneous trajectory similarity measurements and build a new model to calculate the similarity for online accuracy improvements.
- We design and implement an online spatial-temporal similarity search prototype system "*Osprey*" based on a given distributed database for the proof-of-concept.
- We conduct extensive experiments to exploit the heterogeneous trajectory datasets for the evaluation of the performance of *Osprey*. The experimental results show that the latency of the system is less than 4s in most cases, and the accuracy is more than 70% when the similarity exceeds 0.5, both are better than the compared algorithms.

The remainder of the paper is organized as follows. Sect. 2 discusses some related work and Sect. 3 introduces the formulation and challenges of the similarity search

problem. After that, we describe the framework of *Osprey* in Sect. 4 and cover its implementation of in Sect. 5. We present the experimental results are presented in Sect. 6, which is followed by the conclusion and discussion in Sect. 7.

2 Related work

Trajectory similarity search aims at finding from a dataset the trajectories with the highest relevance to a query argument. This procedure typically consists of a definition step and a query processing step [12]. First, a similarity function is defined to evaluate the spatial and temporal similarities between two trajectories, it may involve spatial [13], temporal [14], textual [15], and density elements [2]. Second, an efficient algorithm is developed to retrieve the trajectories which are spatial-temporally close to the query trajectory. Previous works on performing the trajectory similarity search require massive iterative computation and are thus unfriendly to parallelization, such as Dynamic Time Warping (DTW) [6, 16], Longest Common Sub-Sequence (LCSS) [17], Edit Distance on Real Sequence (EDR) [3], and so on.

Some pieces of research use tree structures to speed up the similarity search, such as TSEIT [18] and Proximity Forest [19]. TSEIT is a search tree-based approach that accelerates the data retrieve by storing the time-series envelopes in its nodes. With the design of the tree structure, the performance of TSEIT is significantly improved. Proximity Forest, on the other hand, is an ensemble of proximity trees, which is built on top of decision trees. The experiments demonstrate that the Proximity Forest can not only improve the performance but also achieve high accuracy. However, this algorithm has a poor tolerance to measurement errors and does not perform satisfactorily to time inconsistency and data loss in multi-source heterogeneous trajectories. Kondor et al. [20] conducted an in-depth study on the matching of data sets from different sources, and show that the main determinant of matchability is the expected number of co-occurring records in the two datasets. Therefore, for the heterogeneous search problem, we should pay more attention to increasing the number of matches as much as possible, which contributes to improving the accuracy.

To address the measurement error issue, an intuitive approach is to leverage the method of relaxation. Based on this idea, Pelekis et al. [21] defined two types of similarity, spatial-temporal and (temporally-relaxed) spatial similarity, for trajectory similarity search. As an improvement, Mao et al. [8] presented a segment-based trajectory similarity measure. Comparing with the point-point distance, the proposed segment-segment distance can not only effectively reduce the sensitivity influence of the sampling methods but also substantially improve the accuracy of similarity search. The idea of segmentation is widely used in spatial-temporal data processing systems. It speeds up the range queries and achieves high degree of parallelism by discretizing the spatial and the temporal dimensions. In contrast, DISTIL [22, 23] is a distributed in-memory spatial-temporal data processing system that supports low latency concurrent processing of spatial-temporal range queries by isolating the calculations to the nodes containing the relevant tiles. We are inspired by this work to use a distributed database for parallel similarity search. Unfortunately, the segment-based method cannot address the synchronization issue with respect to the inconsistent acquisition rates

caused by multi-source trajectory efficiently. To address this shortcoming, Sun et al. [24] proposed a spatial and temporal constrained trajectory similarity (STCTS) model for asynchronous multi-source trajectories. In this method, they proposed a concept, called "optimal matching point", and counted it under the spatial and temporal constraints within a threshold range to measure the similarity between trajectories from different sources. Unlike the simple statistical matching points, Hung [25] et al. gave a formulation of club-aware trajectory similarity and a club-aware clustering algorithm to cluster the similar trajectories into groups. This way of clustering the points of multiple trajectories inspired us to propose a cluster-based method to improve the search accuracy.

Although the proposed spatial-temporal cluster algorithm can potentially improve the search accuracy, it also brings a large number of computation workloads. To achieve an online similarity query, parallel computing is an appealing way to improve the performance. Xie et al. [5] described a generic and scalable framework for processing the distributed similarity search from a large set of trajectories. Although their Spark-based framework is impressive with respect to computing capability, its latency is relatively large. Li et al. [26] leveraged the Storm framework and designed a corresponding suffix tree index to speed up response time. Although they did not perform similarity searches, we got some inspiration from the index design. Therefore, to address the same latency issue, Phoenix [27] is used in our *Osprey*, which can conduct a similarity search on a billion-level dataset and reduce the latency for better overall performance.

3 Problem formulation and challenges

This section introduces some related background knowledge and the formulation of the problem in this research. The traditional similarity search problem is defined as follows: given a trajectory Q extracted from data source S, the trajectory similarity search will return all the subsets of trajectories extracted from the same data source, whose *similarity factors* to Q exceeds a pre-defined threshold θ [2, 3]. More formally, to simplify the input query trajectory in similarity search problem, we define it in Def 1.

Definition 1 *(Input Query Trajectory)* $Q = [(x_1, y_1, ts_1, te_1), ..., (x_n, y_n, ts_n, te_n)]$, here (x, y) denotes the position of the spatial plane, and ts and te denote the start time and the end time spend at this point, respectively.

We abstract the spatial-temporal point as a triple (x, y, t) and confirm the unique trajectory with the exact spatial and temporal information of a particular observation point. To identify the unique trajectory, we use a unique tid to mark the movement trace (i.e., a specific trajectory tid of target object). Then, given trajectory dataset $\mathbb{D}$, we can define a trajectory point p in $\mathbb{D}$ as follows:

Definition 2 *(Trajectory Point)* $p = L(tid, x, y, t)$ i.e. $\nexists((tid_1, x_1, y_1, t_1), (tid_2, x_2, y_2, t_2)) \in \mathbb{D} \backslash (x_1 = x_2) \wedge (y_1 = y_2) \wedge (t_1 = t_2)$, here, (x, y) specifies the position of spatial plane, and t represents the collection time.

Based on this definition, the traditional similarity search problem for any query trajectory Q is defined to find the subset $\mathbb{R}$ from $\mathbb{D}$ that satisfies:

$$\mathbb{R} = \{sim(Q, r) > \theta \mid r \in \mathbb{D}\} \tag{1}$$

The main difference between homogeneous and heterogeneous searches depends on whether or not Q and $\mathbb{D}$ come from the same data source S. More importantly, compared to its homogeneous counterpart, the heterogeneous similarity search problem is characterized by its ground truth, which can be used to validate whether Q and the most similar trajectory are traces of the same target (i.e., the accuracy we discussed later). Therefore, the objective of heterogeneous similarity search is to search the most similar trajectory r^*, which is formalized as follows:

$$r^* = \underset{r \in \mathbb{D}}{\operatorname{argmax}}\{sim(Q, r)\} \tag{2}$$

here, sim denotes the similarity measure function, which is often defined as the distance between two trajectories (see Sect. 4.3.1), and Q and rs denote the query trajectory and the trajectory in $\mathbb{D}$, respectively. Although this form of definition is identical to that of the similarity search in traditional cases, they are different in some aspects.

Traditional similarity search usually aims at the data collected by the same measurement, that is, Q and r are homologous or come from the same sensing system. This property ensures that the two sequences can be aligned well even if there is a small measurement errors between them. The similarity can be measured by accumulating the distance between all the pairs of aligned points. However, besides a lot of complicated calculations for distance computation, alignment-based methods also need the two trajectories to have approximately the same number of sampling points. Unfortunately, for the trajectories from different domains (e.g., different sensors), their number of sampling points is not restricted and coupled with the huge amount of data. Consequently, it is hard for alignment-based methods to be adopted in such a scenario.

Besides, when the query data and the trajectories in database come from two completely different collecting systems, e.g., the trajectory of different dimensions collected by different sensors in urban computing, there is no direct interrelation between them. As such, it is even hard, if not impossible, to recognize that the two trajectories are belong to the same object (e.g., face-recognizing data and WIFI connection data). As indicated in our study, the heterogeneity is first manifested in *data formats* as shown in Def. 1 & 2 where the query trajectory is defined by a sequence of 4-element tuples (x, y, ts, te) while the trajectories in database are described by a sequence of triples (x, y, t), say, for example, WiFi connection vs. GPS information.

In addition to the heterogeneity in data format, another is in the *data skewness* as shown in Fig. 1. Due to different sensing systems, one trajectory ("query trajectory") may be very sparse and skewed while the stored trajectories could be relatively detailed and regular. For example, the WiFi connections are not always available evenly in different places along a path while the GPS location information could be available regularly. Given this scenario, it is not only difficult to make pairwise alignments between them for similarity analysis but also not always appropriate to adopt those Euclidean-distance based methods to calculate the similarity as the found paths with

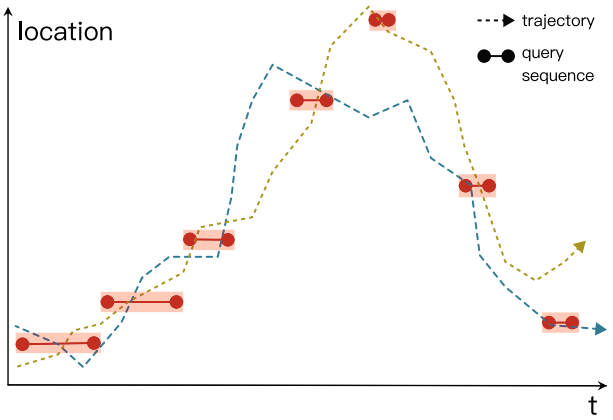


Fig. 1 A case that two types of data come from different sensing systems. WIFI connection data ("query sequence" in the figure) does not pinpoint users' specific location, only the time spent near an acquisition device. In contrast, face recognition ("trajectory" in the figure) can track the trajectory through multiple cameras

small Euclidean distances might not the paths for the query object[1]. Thus, some ways based on probability and set theories need to be integrated for compensations. Such a strategy usually involves the fusion of multiple sensors to validate two trajectories in different dimensions for the same object.

Given these heterogeneity challenges, the traditional similarity search based on the point-wise methods is not always effective anymore to fulfill the accuracy requirements in our settings. Therefore, how to achieve an effective heterogeneous similarity search is a highly desired problem to be solved.

4 Framework

This section introduces the framework of our approach where an extended spatial clustering algorithm is first designed to partition the spatial plane to facilitate the heterogeneous search and then a similarity measurement that combines multiple often-used metrics is presented to improve the search accuracy.

4.1 Overview

As mentioned above, for a multi-sensor collection system, the heterogeneous search suffers from the inherent alignment issue between the query and the stored trajectories due to the time and measurement inconsistency. As such, the traditional distance-based similarity search (i.e., $sim(Q, rs) > \theta$) is not always sufficient. To this end, we propose a new two-phase similarity search framework as shown in Fig. 2.

The first phase is called *Zoom-Out phase* where the spatial plane all data collection points is partitioned through an extended spatial clustering algorithm in *pre-processing*

[1] The paths up and down a viaduct are actually different paths although they have small Euclidean distances.

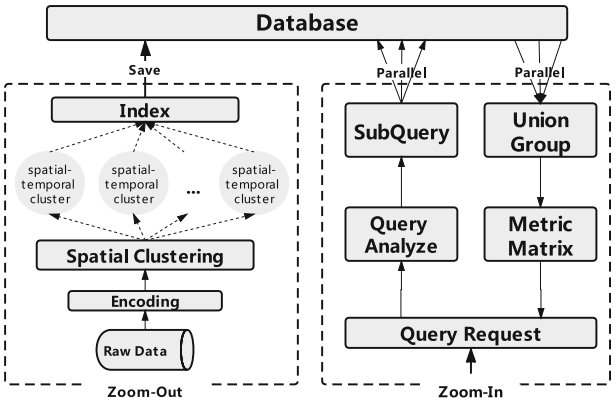


Fig. 2 Two-phase similarity search framework. There are two major parts in this framework: Zoom-Out and Zoom-In, representing the two stages of the search. The two parts interact via a distributed database

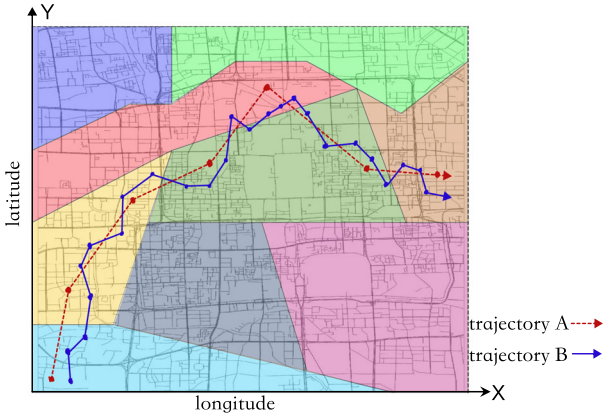


Fig. 3 The space plane is divided into multiple clusters, and different colors represent different clusters. Trajectory A (red dashed line) and B (blue line) represent two different types of trajectories, starting from bottom-left corner to top-right corner (color figure online)

stage into several regions as shown in Fig. 3 where the distance between the different regions after the partition is as large as possible, and the distance between points inside regions is as small as possible. Then, the trajectories are indexed according to the partitioned regions to form *spatial-temporal clusters*.

The second phase is called *Zoom-In phase*, which is performed in the *online query processing stage* where the concept of distribution approximation is used to measure the probability that the same object's movement generates two different trajectories through the intersection of different clusters. In particular, the query trajectory is divided into different parts in both time and space as shown in Fig. 3. As such, we can distribute the similarity calculations into different spatial-temporal clusters, which not only maximizes the parallelism but also aligns different trajectories into groups to ease the similarity calculations. After the sub-trajectory in each cluster has been found, the full similar trajectory is constructed by concatenating them as a whole. To ensure the

high cohesion of spatial-temporal clusters, the trajectories should be partitioned by using clustering algorithms as a pre-condition, which we will discuss in the sequel.

4.2 Spatial clustering algorithm

There are a variety of clustering methods that can be used to partition the trajectories in our settings (e.g. K-Means [28], DBSCAN [29], TAD [30], etc.). These algorithms mainly take into account the Euclidean distance between object positions. However, in most practical scenarios, objects (e.g., commuters and vehicles) often move in a spatial network rather than in a Euclidean space, and thus exhibit some gathering behaviors. So we adopt the spectral clustering algorithm [31] in our case as in general it performs better than traditional clustering methods with respect to highly cohesive datasets.

Spectral clustering is a technique rooted in graph theory, which relies on a *weight matrix*'s eigenstructure to partition points into disjoint clusters in such a way that points in the same cluster have high similarity while points in different clusters have low similarity. Typically, the weight matrix is often derived from a set of pairwise similarities defined by various types of distances between the points to be clustered. However, in our work, instead of using the distance-based similarities between points, we leverage the number of adjacencies defined as $cnt_{(x_a,y_a),(x_b,y_b)}$ in Eq. (3) in historical spatial-temporal trajectories to represent the relationships between spatial point $a(x_a, y_a)$ and $b(x_b, y_b)$,

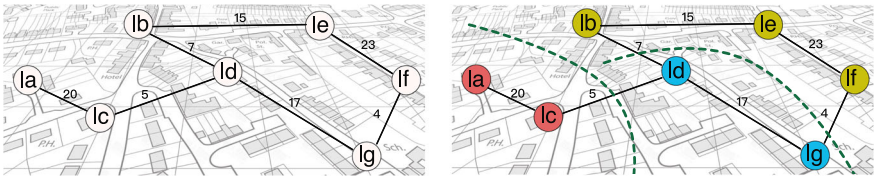
$$cnt_{(x_a,y_a),(x_b,y_b)} = \sum_{tr \in \mathbb{D}} I(tr, (x_a, y_a), (x_b, y_b)) \tag{3}$$

where $\mathbb{D}$ is the trajectory dataset and the number of pair-wise adjacencies for a specific time-ordered trajectory tr of an object is defined as follows:

$$I(tr, (x_a, y_a), (x_b, y_b)) = \begin{cases} i & \text{the number of times that} \\ & a \text{ follows directly after } b \text{ in } tr \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

$cnt_{(x_a,y_a),(x_b,y_b)}$ specifies how many trajectories pass by these points, which is more representative with respect to the interrelations between the points on trajectory as the found paths with small Euclidean distances might not the correct paths for the query object as we discussed. As such, it is a better indicator of the pairwise similarity between the points from the perspective of the trajectory.

To represent the spatial plane by a graph as inputs to spectral clustering algorithm, we abstract the data points on the plane as well as their relationships into vertexes and edges. For the vertex, given the huge number of different points represented by the original latitude and longitude (x, y), it is necessary to encode the adjacent points into the same code, which can not only reduce the amount of calculations but also facilitate the representation of adjacent graphs. To this end, we encode the continuous location (x, y) of the target object with *GeoHash* [32], whose code has a corresponding relationship with GPS(x, y), and precision is related to the encoding length. As such, it is generally enough to select the corresponding precision according to the size of the encoding space.



(a) After encoded, the spatial-temporal points can be regarded as an undirected graph.

(b) The adjacency graph is divided into three categories through spectral clustering, and different colors represent different categories.

Fig. 4 An example of an undirected graph formed by encoding

After the GeoHash encoding, the two-dimensional spatial information is reduced to a single value of "HashID", denoted by l. In this way, the definition of a point can be expressed as an 3-element tuple $L(i, l, t)$, and the trajectory sequence can be sorted along the time axis as follows:

$$P = [(l_1, t_1), (l_2, t_2), ..., (l_n, t_n)]; t_1 < t_2 < \cdots < t_n \tag{5}$$

It is reasonable to assume that adjacency relationships exist between pair of collected points l_a and l_b if the same trajectory ID (i.e., tid) appears successively in l_a and l_b. This relationship can be specified with edge e that connects the pair of vertexes l_a and l_b so that an undirected *collection graph* can be constructed as shown in Fig. 4a, which is the input to the clustering algorithm for the space partition into multiple clusters (regions).

Let all trajectories be transformed into trajectory sequences according to their tids, then we can obtain the adjacency relationships e and vertex v by splitting each trajectory to (l_a, l_b) (l_a, l_b represent all the adjacency "HashID" pairs). Let the weight $w'(l_a, l_b)$ of $e(l_a, l_b)$ be the number of adjacencies between l_a and l_b (i.e., $w'(l_a, l_b) = cnt_{(x_a, y_a), (x_b, y_b)}$ in Eq. (3)). Since it is an undirected graph, the total weight $w(l_a, l_b)$ is set as follows:

$$w(l_a, l_b) = w'(l_a, l_b) + w'(l_b, l_a) \tag{6}$$

The weighted adjacency matrix of the graph $G(V, E, W)$ is the matrix $W = \{w(l_i, l_j) \mid i, j = 1, ..., n\}$, and the degree of vertex $v_i \in V$ is defined as:

$$d_i = \sum_{j=1}^{n} w(l_i, l_j) \tag{7}$$

The degree matrix Deg is defined as the diagonal matrix with the degrees $d_1, ..., d_n$ on the diagonal.

$$Deg = \begin{pmatrix} d_1 & 0 & \cdots & 0 \\ 0 & d_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & d_n \end{pmatrix} \tag{8}$$

The main tools for the spectral clustering are graph *Laplacian* matrices. The non-normalized graph *Laplacian* matrix is defined as:

$$Lap = Deg - W \tag{9}$$

Let $\Lambda = (\lambda_1, \lambda_2..., \lambda_n)$ and λ_i be the eigenvalues of Laplace matrix Lap, and map each trajectory to a lower-dimensional representation based on the corresponding eigenvectors. Finally, the points are assigned to a given number (which is denoted as $n\_cluster$) of classes by the K-Means algorithm based on the new representation.

Algorithm 1 Data pre-processing algorithm

Require: $D(L(i, x, y, t))$: a heterogeneous dataset of $L(i, x, y, t)$; $n\_cluster$: the number of clusters
Ensure: partition table $\mathbb{C}(l, cid)$
1: initialization
2: RD$(l_a, l_b) \leftarrow$ encode raw data by GeoHash and transform into trajectory pair.
3: initialize matrix $W_{n\times n} \leftarrow 0$
4: **for** (**do**each pair (l_a, l_b) in RD)
5: $W_{ab} \leftarrow W_{ab} + 1$
6: $W_{ba} \leftarrow W_{ba} + 1$; Eq.(6)
7: **end for**
8: initialize matrix $Deg_{n\times n} \leftarrow 0$
9: **for** $(i \leftarrow 1...n)$ **do**
10: $d_i \leftarrow 0$
11: **for** $(j \leftarrow 1...n)$ **do**
12: $d_i \leftarrow d_i + w_{ij}$; Eq.(7)
13: **end for**
14: $D_{ii} \leftarrow d_i$
15: **end for**
16: $Lap \leftarrow Deg - W$; Eq.(9)
17: $\Lambda_n \leftarrow$ solve $| Lap - \lambda I | = 0$ extract the eigenvalues $(\lambda_1, \lambda_2, ..., \lambda_n)$
18: initialize matrix $M_{n\times n}$
19: **for** $(i \leftarrow 0...n)$ **do**
20: $v_i \leftarrow$extract eigenvector for λ_i
21: $M_{*i} \leftarrow v_i^T$
22: **end for**
23: $\mathbb{C}(l, cid) \leftarrow$ K-Means$(M, n_cluster)$

The complete algorithm is given in Algorithm 1. At first, the algorithm uses *GeoHash* to encode the raw data spatial information and transforms it into trajectory pairs.(`Line2`). Afterward, a clustering algorithm is used to group the Spaces on the trajectories. Firstly, the weights of the adjacency matrix of the undirected graph are calculated on the sequence (`Line3-7`). Then the degree matrix is calculated (`Line8-15`), and finally the Laplace matrix Lap of the adjacency graph is obtained (`Line16`). Finally, through the eigenvalue decomposition of Laplace matrix Lap (`Line17`), the eigen matrix M of the spatial-temporal sequence can be extracted (`Line18-22`). Then the partition table C of the space can be obtained by using the K-Means method (`Line23`).

A working example of this algorithm is illustrated in Fig.4b where the vertices are clustered according to the generated weighted undirected graph. The id of the cluster

(cid) is used as the track point index. There are more adjacencies between points within each cluster, which means that they have a higher probability of belonging to the same area. They can be considered to be in the same position when calculating the similarity. After the space is partitioned by clustering, a lookup table $\mathbb{C}(l, cid)$ is obtained. With this table, the spatial-temporal trajectories of the raw data can then be processed into spatial-temporal clusters and stored. Usually, the time covered by the spatial-temporal cluster would be abstracted as a range from *starttime* to *endtime*, the trajectory in the same partition can thus be defined as (i, cid, t_s, t_e). However, we still keep it as $[(i, cid, t), (i, cid, t+1), ..., (i, cid, t+k)]; (k = t_e - t_s)$ for the sake of simplification in the calculations.

We are now making an approximate time complexity analysis of Algorithm 1, let m be the number of data records, i the number of tids, and n the number of location points after encoding. For Algorithm 1, the time complexity of `Line2` is $O(m)$. While the time complexities of `Line4-7`, `Line9-15` and `Line17-23` are $O(n^2)$, $O(n^2)$ and $O(n^3)$, respectively. Therefore, the approximate time complexity of Algorithm 1 is $O(n^3 + m)$.

4.3 Query procedure

In the Zoom-In stage, we first introduce the design of the similarity function and illustrate the algorithm of the online query processing afterward.

4.3.1 Similarity function

As mentioned above, a target can be identified by its number of occurrences. However, in most cases, there is no guarantee that a corresponding data collection exists in our observation point. As a result, the occurrence of the target is often equal to or even less than other points for various reasons.

To solve this problem, the general approach is to measure the similarity by performing the spatial-temporal alignment of data and thereby calculating the distance between the aligned points [6, 17]. In contrast, the points in our spatial-temporal cluster have been aligned already, what we need to do is calculating the sum of the similarities for each observation point. Therefore, the questions in our consideration are: (1) interference caused by fixed sensor equipment; (2) large errors of positioning data; (3) serious losses of location data. For the answers, we deliberately exploit three metrics for the similarity measurements: (1) *Self-Information entropy* [33]; (2) *Jaccard index* [34]; (3) *Kullback-Leibler divergence* [35], each one with its own advantages to compensate for each other. We will briefly discuss these metrics with the rationales behind our choices.

a. Self-information entropy

In practice, a trajectory always presents a certain periodicity, and the probability of its tid appearing in a period tends to be stable. For a spatial-temporal similarity search, we need to reduce the influence of some fixed sensor devices on the search results. These noise data from the fixed equipment usually have a large number of trajectories at the same location and a high-frequent collection rate, which could lead to low

information entropy. So the first metric we considered is self-information entropy, which is defined as $h(x) = -\log_2 p(x)$, here, $p(x)$ denotes the probability of tid appearing in the current period. Let the frequency of data collection be s, the period of time be T, and the number of times a trajectory tid has been collected be $count(tid)$, then $p(x)$ can be calculated as:

$$p(x) = \frac{count(tid)}{T/s} \tag{10}$$

To make a meaningful comparison between different metrics, we leverage the z-score algorithm to standardize the metric into a dimensionless form. The definition of the z-score as follows:

$$ZScore(X) = \frac{X - E[X]}{\delta(X)} \tag{11}$$

here, $E[X]$ denotes the the expectation of X, $\delta(X)$ represents the standard deviation of X. Therefore, combining with Eq. (11), we can get the definition of the self-information entropy metric $bits$ as follows:

$$bits = ZScore(-\log_2 p(x)) \tag{12}$$

b. Jaccard index

The self-information entropy reflects the characteristics of trajectory itself, but it cannot measure the similarity between two identical clusters and the query sequence. Thus, we are advocated to introduce *Jaccard index* [34] to measure the similarity of each spatial-temporal cluster. The Jaccard index is a statistic used for gauging the similarity and diversity of sample sets.

We define cnt_i as the number of tid occurrences in the same spatial-temporal cluster, cnt_e as the number of tid occurrences in the time interval $[t_s, t_e]$. Combined with the definition of the input sequence Q in Def. 1, the Jaccard coefficient formula is:

$$J = \frac{cnt_e}{\frac{te-ts}{s} + cnt_i - cnt_e} \tag{13}$$

Similarly, Eq. (11) is used for standardization, and the definition of indicator $jaccard$ is obtained as follows:

$$jaccard = ZScore(J) \tag{14}$$

c. Kullback–Leibler divergence

The Jaccard index measures the similarity of a single spatial-temporal cluster while Kullback-Leibler (KL) divergence [35] is a measure of how one probability distribution is different from a second, reference probability distribution, which is defined as follows:

$$D_{KL}(p||q) = \sum_{i=1}^{N} p(x_i) \times (\log_2 p(x_i) - \log_2 q(x_i)) \tag{15}$$

First, let's define ds as the the number of times of input sequence Q and ds_i as the the number of times of input sequence on each spatial-temporal cluster, then the distribution of input sequence can be obtained as $p(x_i) = ds_i/ds$.

Then, given the complexity of calculating the trajectory tid distribution in each spatial-temporal cluster, we simplify the distribution of each tid to the distribution of tid in spatial-temporal clusters of the input sequence. Define the number of times an tid appears in all the spatial-temporal clusters as es, and the number of times of an tid appears in each spatial-temporal cluster as es_i, so the distribution $q(x)$ of an tid is defined as $q(x_i) = es_i/es$.

Finally, suppose the number of spatial-temporal clusters appearing in Q is N_q, the complete calculation formula of KL divergence can be obtained as follow:

$$kld = ZScore(D_{KL}(p||q)_{N_q}) \tag{16}$$

By leveraging the standardization and the weighted average method, we can calculate the similarity matrix of the trajectories, which can be used to measure the similarity and rank these trajectories. Rather than using one distance metric, the three metrics are more robust to noise and more tolerant of measuring errors. The combine model for features is described in the next section.

4.3.2 Online query processing

In the online query processing part, a query sequence is given to search similar trajectories in the database. For efficient data retrieval and similarity matrix calculation, we propose a query processing algorithm to decompose the query and calculate the foregoing metrics in parallel, as illustrated in Fig. 5.

At the beginning of query processing, we also need to lookup cid from lookup table $\mathbb{C}(l, cid)$ for each $L(x, y)$. The whole sequence from different sources can be decomposed into sub-queries made to each cluster, and aggregated by following *union* operation. Finally, the metrics is calculated and the feature matrix is generated as shown in Algorithm 2.

In Algorithm 2, each spatial-temporal segment of the query sequence is determined its cid by looking up $\mathbb{C}(l, cid)$ and decomposed into sub-queries firstly (`Line1-6`), and the resultset is queried in the database in parallel (`Line7-8`). Finally, according to the design of similarity index: $entropy$, $Jaccard$ and KL are counted on the resultset, and the feature matrix $F_{n\times 4}$ is generated (`Line9-12`), which is defined as follows:

$$F_{n\times 4} = \{(tid(i), bits(i), jaccard(i), kld(i)) \mid i = 1, ..., n\} \tag{17}$$

After getting feature matrix $F_{n\times 4}$, we calculate the similarity according to it. The feature matrix is a high-dimensional representation of the similarity matrix, which indicates three-dimensional preferences for similarity search, and different data sets have different sensitivity to each metric. Our goal is to combine this high-dimensional representation into a similarity, which can be regarded as a variant of the multi-objective optimization problems (MOPs). The weighted sum method is a typical method to solve this problem, which converts multi-objective into a single-objective optimization through a linear combination [36]. Meanwhile, the coefficients are specific to the dataset, and we can learn these coefficients according to the properties of dataset and user's favorites. To this end, we add weight coefficients: α, β, γ

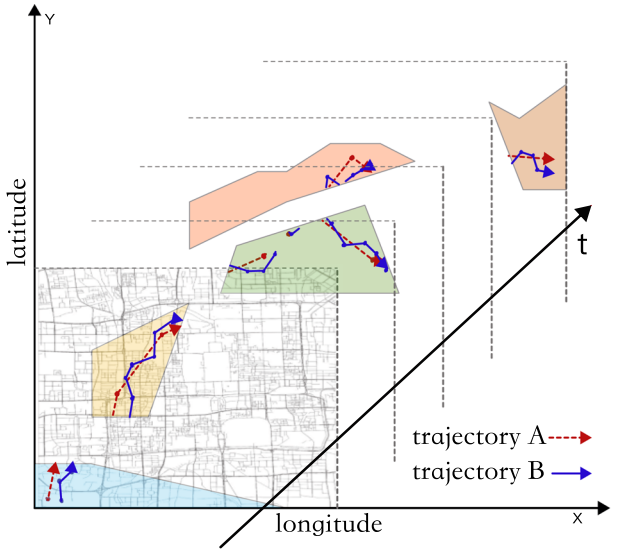


Fig. 5 Decomposing the query to calculate the similarity. Deconstruct the example in Fig.3 into five parts to computing the similarity in parallel

$(\alpha, \beta, \gamma \in [1, 0); \alpha + \beta + \gamma = 1.0)$ to the metrics that are already dimensionless, the final similarity formula is as follows:

$$similarity = \alpha \times bits + \beta \times jaccard + \gamma \times kld \tag{18}$$

where the three indicators measure the property of intra-trajectories, inter-relationships between spatial-temporal clusters, and the property of inter-trajectories, respectively. Therefore, the three respective coefficients are used to control the preference degree of the similarity search. For example, if some noise data with fixed positions need to be filtered out, α needs to be appropriately increased while if those trajectories staying at the same place as long as possible need to be found out, a larger value of β should be taken into consideration. Finally, if the trajectories that occur together at multiple different locations are desired, a higher value of γ will improve the accuracy.

Finally, the resultset is ranked by the similarities according to these parameters (`Line13-14`). Let k be the number of trajectories in the resultset, v be the average number of points of each trajectory. It's obvious that `Line1-6` takes order K time, and `Line9-13` only needs to traverse all the resultset linear times to calculate the three metrics with a time complexity $O(kv)$, and the time complexity of the sorting in `Line15` is $O(k \log k)$. Overall, the approximate time complexity of Algorithm 2 is $O(k(\log k + v))$.

Algorithm 2 Online query process

Require: $Q = [(x_1, y_1, ts_1, te_1), ..., (x_n, y_n, ts_n, te_n)]$
Ensure: trajectory r^*
1: **for** (each $L(x, y, ts, te)$ in Seq) **do**
2: $l \leftarrow$GeoHash(x, y)
3: $cid \leftarrow$ lookup clusterID for l in $\mathbb{C}(l, cid)$
4: $(tis, tie) \leftarrow (MaxValue - L(ts), MaxValue - L(te))$
5: subsqls $\leftarrow$ subsqls + "select * from table between cid+tis and cid+tie"
6: subsqls $\leftarrow$ subsqls + "union all"
7: **end for**
8: sql $\leftarrow$ "select agg(metrics) from "+subsqls+"group by id,cid"
9: resultset $\leftarrow$ execute sql in phoenix
10: $bits \leftarrow$ compute the entropy for resultset; Eq.(12)
11: $jaccard \leftarrow$ compute the Jaccard index for resultset; Eq.(14)
12: $kld \leftarrow$ compute the KL divergence for resultset; Eq.(16)
13: $F_{n\times 4} \leftarrow (tid, bits, jaccard, kld)$
14: $Sim[tid, similarity] \leftarrow [F[tid]; [\alpha, \beta, \gamma] \times F[bits, jaccard, kld]^T]$; Eq.(18)
15: $r^* \leftarrow$ Top1(ranking resultset with Sim)

5 Framework implementation

5.1 Overview

To prove the concept of the proposed two-phase framework for heterogeneous similarity search, we prototype an online system, called "*Osprey*", to improve the accuracy and latency of the search. In *Osprey*, the raw data are partitioned into spatial-temporal clusters in the Zoom-Out phase, and the corresponding metrics are designed for similarity ranking in the Zoom-In phase.

This section provides an overview of the *Osprey* design, which is composed of two major components: a data pre-processing component and a query processing component as shown in Fig. 2. In the beginning, the raw data are formatted and indexed by the pre-processing component and stored in a database. After the pre-processing is performed, the query processing can accomplish the online similarity query to these stored data.

In the Zoom-Out phase, *Osprey* first applies the modified clustering algorithm as described in Sect. 4.2 to the historical data, and then format the raw data and store them into a database. While in the Zoom-In part (Sect. 4.3.1), the similarity function with three metrics as well as a query processing is implemented.

Since the use of the clustering algorithm with high complexity and the spatial-temporal clusters, both of them result in a large amount of computations. To achieve the online response in large-scale datasets, the prototype is implemented in a cluster architecture to improve the degree of parallelism and accelerate the similarity search.

5.2 Implementation

In this section, we describe how to implement *Osprey* in a cluster with emphasis on maximizing the degree of parallelisms to improve the efficiency of heterogeneous

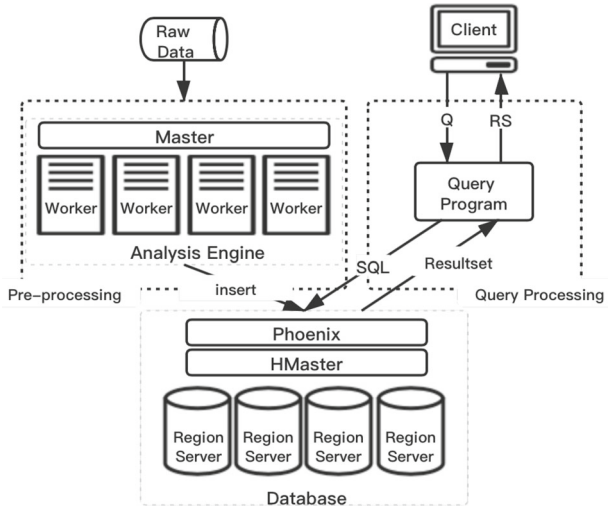


Fig. 6 The implementation of *Osprey*, a Spark-based analytics engine is used in the pre-processing part, and the trajectories are saved into a database named Phoniex, which based on HBase

similarity search. To this end, we organize *Osprey* into several physical modules as illustrated in Fig. 6 where an analysis engine is implemented in the data pre-processing component, which could format and store data to a database, and the query processing, running at the query processing part, is implemented between the database and client.

For the data pre-processing component, a Spark-based analytics engine is used for data pre-processing. We use a module, called *Spark-SQL*, which can connect to the database and perform data reading and writing directly, and the raw data are stored in *Parquet* format and processed in memory in the form of *dataframe*.

In order to implement an online search system, we deliberately choose a distributed database system (DDBS) as the storage substrate. At present, there are many popular distributed storage, such as HBase [37], Greenplum [38], InfluxDB [39], etc.. After comprehensive comparison of their performance and scenario supports, we adopt a SQL on HBase architecture for the data storage in our prototype. Among various plugins for SQL on HBase, we select Phoenix [27], which is widely used to sufficiently support for SQL, as our implementation framework. For HBase and Phoenix, the design of rowkey is crucial because the fastest conditional range query in HBase is a rowkey-based scan operation. Therefore, the design of rowkey includes all the conditions to be queried.

Considering that the partitioned spatial plane is invariant and the temporal range is evolved, we put the spatial information in front of the temporal information. Moreover, the most recent data is usually those that are most likely to be queried, so the time instances in reverse order are designed as the index of the temporal information. Let $tindex = MaxValue - Timestamp$ ($MaxValue$ is the maximum whose number of digits is identical to the timestamp). Finally, to avoid the duplicates of rowkey, it's necessary to concatenate tid at the end of the rowkey, i.e., $RowKey = cid+tindex+tid$.

Table 1 Database table fields of the trajectories saved in Phoenix

Field	Description
rk	Rowkey
cid	Cluster ID
loc_x	Location on x
loc_y	Location on y
ts	Collected time
tid	Trajectory ID
$bits$	Self-information entropy
cnt_e	The number of tid occurrences in a query spatial-temporal cluster

In summary, the database fields include rowkey and cid. In general, the data is not evenly distributed across the space, so a *hotspot* problem is inevitable if cid is used as the prefix of rowkey. To solve this problem and make the calculation in parallel, the rowkeys are scattered as widely as possible. Since the Phoenix is used as the data storage, we can use SQL for data retrieval. The final fields of the database table in our implementation are listed in Table 1.

6 Empirical studies

In this section, we conducted empirical studies to evaluate the *Osprey* framework in terms of its accuracy (we defined it in Sect. 6.2) and latency (the time it takes to get a response after a query is submitted). First, we introduced the experimental setups and overviewed the profile of the selected trace data, then we designed experiments to study the effects of the proposed cluster algorithm, followed by evaluating its accuracy and analyzing the results. After that, we evaluated the latency of *Osprey* to show how the online performance is achieved. As our work is the first to study the heterogeneous search based on city metro WiFi positioning data and metro WiFi connection data, it lacks the exact studies as references for fair comparison. Thus, in these experiments, we focused squarely on the proposed approaches in terms of its characteristics *per se*. Our empirical study results show that *Osprey* has high accuracy about more than 0.7 in heterogeneous similarity search with quick response (less than 4s in most cases) for large-scale datasets.

6.1 Experimental setups

6.1.1 Testbed configurations

We set up a testbed based on the *Osprey* prototype as an 8-server cluster, each being configured with 8 cores Xeon E5-2620 v4 processors and 32 GB DRAM, and interconnected by a 10Gbps network. Among the 8 servers, we installed Spark2.0 on three of them, with two workers and one master, to perform the pre-processing phase.

On the other hand, we implemented the query program in Scala2.11, deployed on a client server, and set up a restful HTTP interface to submit the test query. Finally, we installed Phoenix on HBase1.3 with another three *Region Servers* and one *Hmaster* on the remaining four servers. For the dataset distribution, we also set parameter SALT_BUCKET in HBase as 20, which means partitioning the dataset based on 20 regions, a reasonable value for our testbed.

Again, the preference of similarity on different indicators depends on dataset characteristics and user's favorites. And we can learn the coefficients from historical trace data for certain scenarios to improve the accuracy, but this is beyond the scope of this paper. As a typical case without any preference for the indicators as in our settings, we gave equal weights to each metric.

6.1.2 Trace data profile

We use two datasets in the experiment to verify the model. A publicly available small-scale dataset (1) is used to validate the accuracy of *Osprey* in the first experiment and compared with other baseline algorithms (K-Means, DBSCAN). To simulate heterogeneous search, we randomly sample trajectories from the dataset and add some noise as the query sequence; Another large-scale dataset (2) is used to evaluate the performance of *Osprey*. It contains two types of data collected from two different sensor systems: the smaller-scale WIFI connection data is used as the query set, and the WIFI position data is stored in *Osprey*.

(1) *T-Drive trajectory sample data* [40, 41]: This is a sample of T-Drive trajectory dataset that contains a one-week trajectories of 10, 357 taxis in Beijing, China. The total number of points in this dataset is about 15 million and the total distance of the trajectories reaches 9 million kilometers. The format of dataset $\mathbb{D}$ is in a form of $L(taxiId, x, y, t)$, which is consistent with our previous definition in Definition 2.

(2) *WIFI position data:* We used the trace data collected by a WIFI positioning system that locates the MACs of mobile devices through the WIFIs in the metro stations of Shenzhen, metropolitan city in south China [42]. The format of dataset $\mathbb{D}$ is in a form of $L(MAC, x, y, t)$. The collected records an average of 24 million per day, and we used a dataset of half a year (from 2018/09/01 to 2019/03/31) in the experiments, with a total number of 4.3 billion records. The total number of different MACs is about 22 million, and the average number of MACs is 3.2 millions per day.

6.2 Similarity model accuracy

The accuracy of the similarity model could be affected by many factors. We are particularly interested in the cor-relationships between the similarity accuracy and the proposed clustering algorithm. To this end, we designed an experiment to evaluate the accuracy of *Osprey*.

We firstly applied different clustering algorithms to conduct spatial discretization of T-Drive data whereby the influences of K-Means, DBSCAN, and the spectral clustering algorithm we developed are compared. Also, we validated the improved similarity with respect to the proposed metrics. Fig. 7 illustrates how the space is visualized

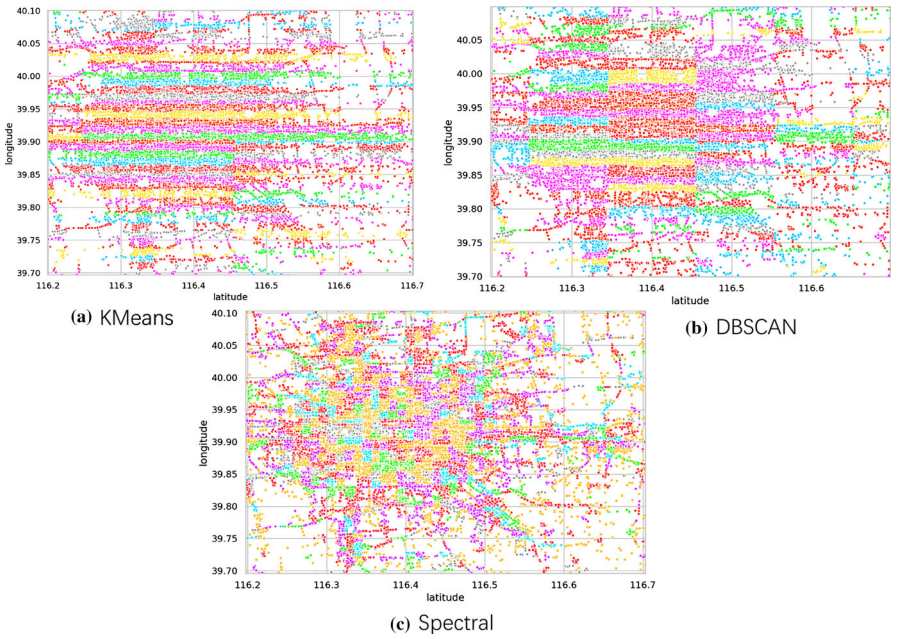


Fig. 7 Different clustering algorithms discretize the space. **a** DBSCAN; **b** K-Means; **c** Spectral Cluster (ours); Different colors represent different spatial clusters with a unique cluster ID

after discretized by different clustering algorithms. The adjacency-based method is more uniform across different clusters, and thus in line with the taxi trajectory of the real-world than the other two algorithms.

Afterward, we constructed a test dataset by deliberately adding some noise to the originally gathered data to mimic the measurement errors of different acquisition sources. To this end, original data point (t, x, y) is revised with a uniform random offset as $(t + random(-20mins, 20mins), x + random(-0.5, 0.5), y + random(-0.5, 0.5))$. Then, we randomly selected 900 trajectories as the test dataset and sampled m ($m \in [2, 20]$) collection points on each trajectory as the input to verify whether the tid found by the similarity search is the same or not as the input. For the sufficiency of the experiment, we further conducted a total of 6 independent test groups. Each group contains 900 different query trajectories, and the distribution of the trajectory size in each group is shown in the boxplots in Fig.8, where Group5 increases the size of input trajectories while Group4 and 6 reduce it.

We modified two advanced algorithms—BDS [4] and TPDC [12]—as baselines to evaluate Osprey. The two algorithms are designed to compute the similarity of the trajectory sets in an offline way, their latencies are much greater than that of the near real-time *Osprey* (usually more than 200s on the million-scale dataset). Hence, these two algorithms are not comparable to the proposed framework in terms of system overhead. We simplified these two algorithms with focus on using the similarity function of the two algorithms for the heterogeneous similarity search. Notably, the search goal is also to find the most similar trajectory to the query trajectory in different domains.

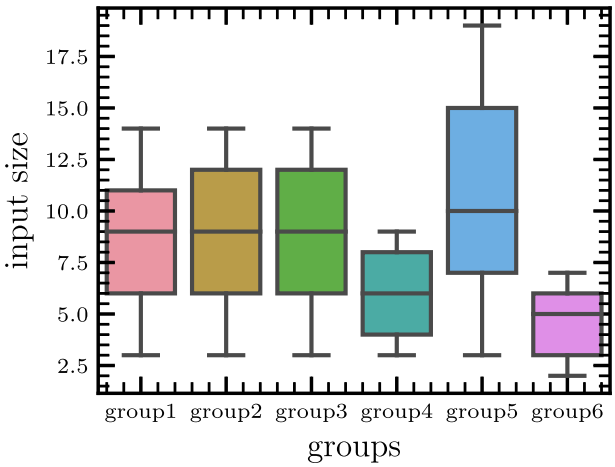


Fig. 8 The size distributions of input query trajectories in each group

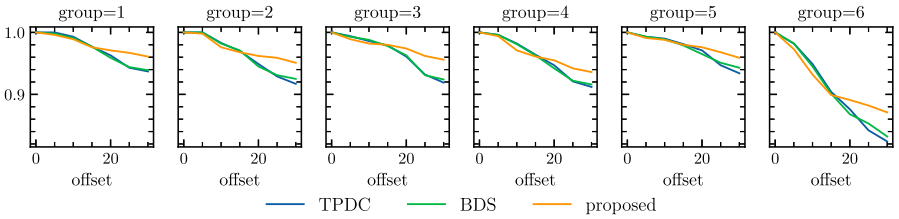


Fig. 9 Comparison of accuracy between *Osprey* and the two modified algorithms. The x-axis of each figure is the offset of random deviation in time, and the y-axis is the accuracy

To investigate the impact of the time error caused by heterogeneity on the search accuracy, we set the random offset range of time from 0 minute to 30 minutes, a reasonable range to cover the duration of one taxi trip. We compared the accuracy of the these algorithms whose results are shown in Fig. 9. One can see that the accuracies of all the algorithms are roughly the same when the minor error in time, while all the accuracies decrease as the random offset of time increases, whereby the reduction of Osprey's accuracy is more moderate than others. The results show that Osprey achieves higher accuracy in case of a significant error on time offset than the other compared algorithms, which indicates it is more applicable for the heterogeneous similarity search scenarios.

The results of the another comparison experiment are shown in Fig. 10 where the accuracy of the three compared algorithms—the proposed spectral clustering, DBSCAN and K-Means are compared. From these test groups, one can observe that the proposed spectral clustering is the best in accuracy, which is better than DBSCAN about 2%, and K-Means about 3%, respectively. When the number of sampling points increases, the corresponding accuracy will also be improved accordingly, and the gap between the spectral clustering and the other two algorithms become smaller (Group5). This is not surprisingly as when the input samples are increasingly sufficient,

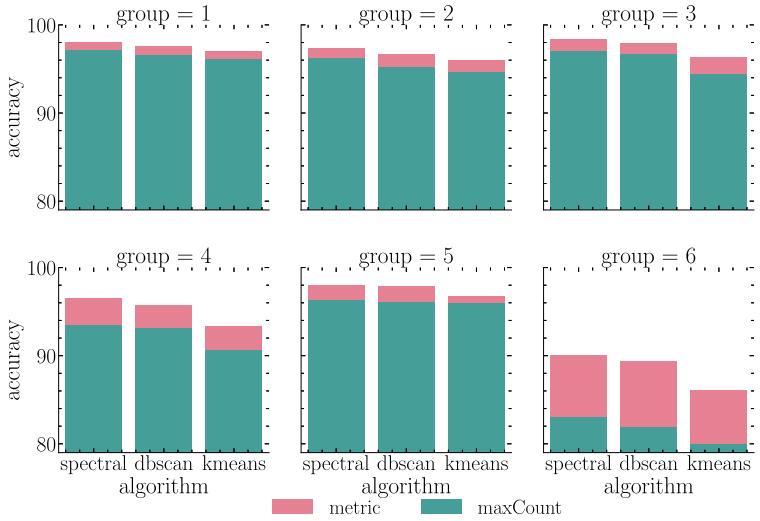
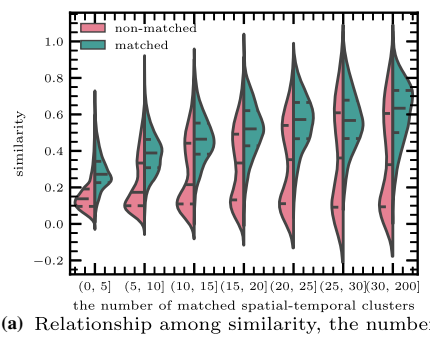


Fig. 10 Accuracy comparison between Spectral clsutering, DBSCAN, and K-Means. The higher bar chart (red one) is using our proposed metrics

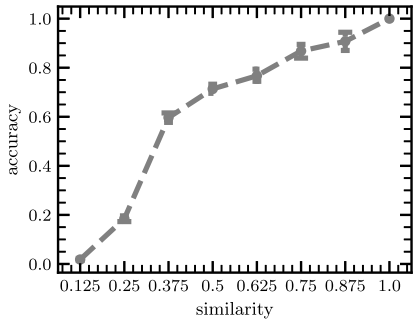
the information will gradually become rich to achieve high accuracy, which would relatively weaken the effects of algorithm optimization.

In Group4 and 6, When fewer sampling points are input, the corresponding accuracy decreases. In this case, our method exhibits more relatively obvious advantages, the accuracy is about 5% higher than that of K-Means. Meanwhile, these groups show that our propose metrics could effectively improve the accuracy over simple method (maximum number of occurrences), with the improvements increasing from 1 to 9% as the number of sample points decreases, which means the improvement can be increased even more for fewer data. This is because the metrics we proposed for this case can effectively score and sort the trajectories with the same occurrences in the results, and select the most similar trajectory. The experiment shows that the accuracy of our method is not only better than that of the compared methods, but also effective to improve the accuracy of similarity search with less data, and has better robustness for the situation when the data is in shortage.

In the other experiment, we first obtained the trajectories of targets based on some metro WiFi positioning method, and then made samples from the sequences obtained by an WIFI connection system as input to verify whether the tid (i.e., MAC address) with the highest similarity is the target. To this end, we collected 13, 629 test data points from the WIFI connection system. Given the collection rates are different between these two WiFi devices, and also the information gathered by each individual device is divergent, the distribution of data collected by WIFI connection system would be quite different from that by WIFI positioning system. As a result, a large number of data items collected by the WIFI connection system are possible not collected by the WIFI positioning system, reflecting the heterogeneity of the data sources.



(a) Relationship among similarity, the number of spatial-temporal clusters and correctness.



(b) Relationship between accuracy and similarity.

Fig. 11 Correlation between similarity and accuracy

Figure 11a illustrates that how the similarity distribution of the matched trajectories is higher than that of the non-matched trajectories when a query trajectory is given, also how the similarity increases with the increase of the number of spatial-temporal clusters. The results not only indicate that it is feasible to identify the target tid based on our similarity search, but also reveal that the current heterogeneous similarity depends on the number of the input clusters.

Additionally, we also evaluated the relationship between similarity and accuracy as shown in Fig. 11b where one can see (from the broken line diagram) that with the improvements of the similarity, the accuracy increases significantly as well, indicating that the accuracy of our heterogeneous similarity model can be improved by providing more multi-spatial-temporal clusters.

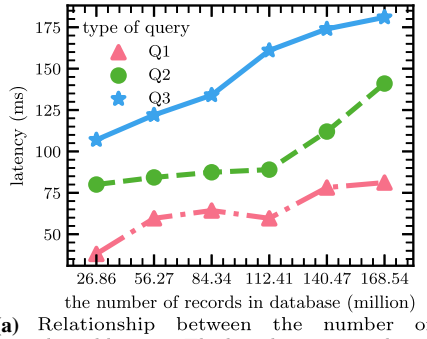
6.3 System overhead

As the similarity search is often performed online, low latency is always highly desired. In this section, we conducted experiments to evaluate the query processing in *Osprey*. By adjusting different input data sizes, the overhead of *Osprey* in term of latency and the accuracy of the algorithm can be evaluated. To this end, we designed three queries to study the relationships between the latency of retrieving trajectory data and the amount of stored data. We sampled three sequences from a trajectory as input, and the time range of each spatial-temporal cluster is selected as 10 minutes while the number of spatial-temporal clusters is configured by 2, 5, and 10, and named Q1, Q2, and Q3 respectively. The number of input query clusters and the corresponding number of the result sets are shown in Table 2.

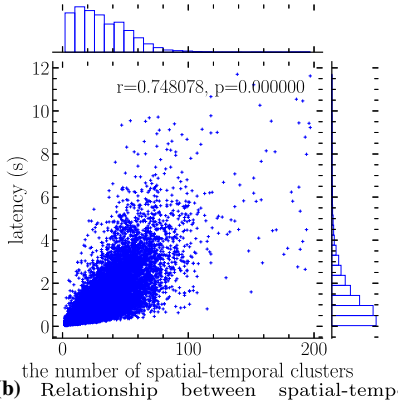
As shown in Fig. 12a, there is a linear relationship between the latency and the number of the query result sets as well as the number of records stored in the database. With the increase of the amount of data, the computation of traditional similarity search quadratically grows with respect to the number of records as they need pairwise comparison between the records. However, given the similarity search in parallel, the computation of *Osprey* is more efficient since as the number of records grows up, its latency increases linearly at large. Since the spatial-temporal data will be increasing

Table 2 The number of the clusters and resultsets

Query type	Clusters	Resultsets
Q1	2	438
Q2	5	3792
Q3	10	6487



(a) Relationship between the number of records and latency. The line diagrams with triangle, circle and star denote three queries: Q1, Q2 and Q3 respectively.



(b) Relationship between spatial-temporal cluster size and latency.

Fig. 12 Comparison of system overhead

continuously with the duration of the system running time, the property of linear growth is valuable for the application of *Osprey*.

After that, we inserted all the data into the database–a total of 4.3 billion items. We deployed the similarity calculation model, so the response time increased slightly. In order to introduce more interference to study the performance of *Osprey* in response time and analyze the accuracy of the similarity model, we varied the time range in $[observedpoints - 10\,min, observedpoints + 10min]$, implying a spatial-temporal cluster lasting 20 minutes.

The scatter diagram in Fig. 12b shows that the latency is approximately linear ($pearson = 0.75$) increase with the number of input clusters, and most of the latency is within $4s$. As such, we can conclude that the *Osprey* design effectively improves the computing performance of the system for large-scale datasets, which as a result meets the requirements for a online search system.

7 Conclusions

In this paper, we designed the spatial-temporal cluster through a spectral clustering algorithm, which samples the adjacency based on historical trajectories, to improve the latency and accuracy in the heterogeneous similarity search. Afterward, we built a model that combines three designed metrics to measure the similarity of spatial-temporal clusters rather than using traditional distance metrics to improve the accuracy in the online query processing phase. Finally, to accelerate the computation and vali-

date the latency and accuracy of our concept, we implemented a system, called *Osprey*, based on a distributed database and investigated its performance through extensive experiments on billions of multi-source heterogeneous trajectories.

Osprey is a heterogeneous similarity search framework designed for large-scale data sets, which provides low-latency and high-accuracy similarity search services. By using set theory and probability theory, Osprey can perform well with sufficient historical data. However, in the case of data at small-scale, the proposed method barely attains more precision similarity than the fine-grained alignment distance methods, which could be another topic deserving our more studies.

Acknowledgements This work is supported in part by Key-Area Research and Development Program of Guangdong Province (No. 2020B010164002), and National Natural Science Foundation of China (No. 61672513).

References

1. Chen R, Jankovic F, Marinsek N, Foschini L, Kourtis L, Signorini A, Pugh M, Shen J, Yaari R, Maljkovic V et al. (2019) Developing measures of cognitive impairment in the real world from consumer-grade multimodal sensor streams. In: Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining, pp. 2145–2155
2. Shang S, Chen L, Jensen CS, Wen J-R, Kalnis P (2017) Searching trajectories by regions of interest. IEEE Trans Knowl Data Eng 29(7):1549–1562. https://doi.org/10.1109/TKDE.2017.2685504
3. Chen L, Özsu MT, Oria V (2005) Robust and fast similarity search for moving object trajectories. In: Proceedings of the 2005 ACM SIGMOD international conference on management of data. SIGMOD '05, pp. 491–502. Association for computing machinery, New York, NY, USA. https://doi.org/10.1145/1066157.1066213
4. Ta N, Li G, Xie Y, Li C, Hao S, Feng J (2017) Signature-based trajectory similarity join. IEEE Trans Knowl Data Eng 29(4):870–883. https://doi.org/10.1109/TKDE.2017.2651821
5. Xie D, Li F, Phillips JM (2017) Distributed trajectory similarity search. In: VLDB 10:1478–1489
6. Ying R, Pan J, Fox K, Agarwal PK (2016) A simple efficient approximation algorithm for dynamic time warping. In: Proceedings of the 24th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems. SIGSPACIAL '16. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/2996913.2996954
7. Ranu SPD, Telang AD, Deshpande P, Raghavan S (2015) Indexing and matching trajectories under inconsistent sampling rates. In: 2015 IEEE 31st International conference on data engineering, pp. 999–1010. https://doi.org/10.1109/ICDE.2015.7113351
8. Mao Y, Zhong H, Xiao X, Li X (2017) A segment-based trajectory similarity measure in the urban transportation systems. Sensors 17(3):524
9. Li X, Zhao K, Cong G, Jensen CS, Wei W (2018) Deep representation learning for trajectory similarity computation. In: 2018 IEEE 34th International conference on data engineering (ICDE), pp. 617–628. IEEE
10. Shang S, Chen L, Jensen CS, Wen J-R, Kalnis P (2017) Searching trajectories by regions of interest. IEEE Trans Knowl Data Eng 29(7):1549–1562
11. Zhang L, Zhao L, Wang Z, Liu J (2017) Wifi networks in metropolises: from access point and user perspectives. IEEE Communicat Magaz 55(5):42–48
12. Shang S, Chen L, Wei Z, Jensen CS, Zheng K, Kalnis P (2017) Trajectory similarity join in spatial networks. Proc. VLDB Endow. 10(11), 1178–1189. https://doi.org/10.14778/3137628.3137630
13. Zheng Y, Zhang L, Ma Z, Xie X, Ma WY (2011) Recommending friends and locations based on individual location history. ACM Trans Web. https://doi.org/10.1145/1921591.1921596
14. Shang S, Ding R, Zheng K, Jensen CS, Kalnis P, Zhou X (2014) Personalized trajectory matching in spatial networks. VLDB J 23(3):449–468. https://doi.org/10.1007/s00778-013-0331-0

15. Zheng, K., Yang, Y., Shang, S., Yuan, N.J.: Towards efficient search for activity trajectories. In: Proceedings of the 2013 IEEE international conference on data engineering (ICDE 2013). ICDE '13, pp. 230–241. IEEE Computer Society, USA (2013)
16. Keogh E, Ratanamahatana CA (2005) Exact indexing of dynamic time warping. Knowl Info Sys 7(3):358–386
17. Vlachos M, Kollios G, Gunopulos D (2002) Discovering similar multidimensional trajectories. In: Proceedings 18th International conference on data engineering, pp. 673–684. https://doi.org/10.1109/ICDE.2002.994784
18. Willkomm J, Bettinger J, Schäler MBöhm K (2019) Efficient interval-focused similarity search under dynamic time warping. ACM International conference proceeding series, 130–139
19. Lucas B, Shifaz A, Pelletier C, O'Neill L, Zaidi N, Goethals B, Petitjean F, Webb GI (2019) Proximity Forest: an effective and scalable distance-based classifier for time series. Data Min Knowl Discov 33(3):607–635
20. Kondor D, Hashemian B, de Montjoye Y-A, Ratti C (2020) Towards matching user mobility traces in large-scale datasets. IEEE Trans Big Data 6(4):714–726
21. Pelekis N, Kopanakis I, Marketos G, Ntoutsi I, Andrienko G, Theodoridis Y (2007) Similarity search in trajectory databases. In: 14th International symposium on temporal representation and reasoning (TIME'07), pp. 129–140. IEEE
22. Patrou M, Alam MM, Memarzia P, Ray S, Bhavsar VC, Kent KB, Dueck GW (2018) DISTIL: A distributed in-memory data processing system for location-based services. GIS: Proceedings of the ACM international symposium on advances in geographic information systems, 496–499
23. Memarzia P, Patrou M, Alam MM, Ray S, Bhavsar VC, Kent KB (2019) Toward efficient processing of spatio-temporal workloads in a distributed in-memory system, 118–127. IEEE
24. Sun L, Zhou W (2017) A multi-source trajectory correlation algorithm based on spatial-temporal similarity. In: 2017 20th International conference on information fusion (Fusion), pp. 1–7. IEEE
25. Hung C-C, Peng W-C, Lee W-C (2015) Clustering and aggregating clues of trajectories for mining trajectory patterns and routes. VLDB J 24(2):169–192
26. Li R, Ruan S, Bao J, Li Y, Wu Y, Hong L, Zheng Y (2020) Efficient path query processing over massive trajectories on the cloud. IEEE Trans Big Data 6(1):66–79
27. Apache: phoenix. [EB/OL]. https://phoenix.apache.org (2020)
28. Gupta S, Kumar R, Lu K, Moseley B, Vassilvitskii S (2017) Local search methods for k-means with outliers. Proceed VLDB Endowm 10(7):757–768
29. Schubert E, Sander J, Ester M, Kriegel HP, Xu X (2017) Dbscan revisited, revisited: why and how you should (still) use dbscan. ACM Trans Datab Sys (TODS) 42(3):1–21
30. Yang Y, Cai J, Yang H, Zhang J, Zhao X (2020) TAD: a trajectory clustering algorithm based on spatial-temporal density analysis. Expert Sys Appl. https://doi.org/10.1016/j.eswa.2019.112846
31. Von Luxburg U (2007) A tutorial on spectral clustering. Statist Comput 17(4):395–416
32. Guo N, Xiong W, Wu Y, Chen L, Jing N (2019) A geographic meshing and coding method based on adaptive hilbert-geohash. IEEE Access 7:39815–39825
33. Wang C, Huang Y, Shao M, Hu Q, Chen D (2019) Feature selection based on neighborhood self-information. IEEE Trans Cybern 50(9):4031–4042
34. Bag S, Kumar SK, Tiwari MK (2019) An efficient recommendation generation using relevant jaccard similarity. Info Sci 483(1):53–64
35. de Matthews AGG., Hensman J, Turner R, Ghahramani Z (2016) On sparse variational methods and the kullback-leibler divergence between stochastic processes. In: Artificial Intelligence and Statistics, pp. 231–239 PMLR
36. Xu H, Zeng W, Zhang D, Zeng X (2019) Moea/hd: a multiobjective evolutionary algorithm based on hierarchical decomposition. IEEE Trans Cyber 49(2):517–526. https://doi.org/10.1109/TCYB.2017.2779450
37. Apache: HBase. [EB/OL]. https://hbase.apache.org/ (2020)
38. Arnold J, Glavic B, Raicu I (2019) A high-performance distributed relational database system for scalable OLAP processing. IPDPS, 738–748
39. InfluxData: InfluxDB. https://www.influxdata.com/products/ (2020)
40. Yuan J, Zheng Y, Zhang C, Xie W, Xie X, Sun G, Huang Y (2010) T-drive: driving directions based on taxi trajectories. In: Proceedings of the 18th SIGSPATIAL International conference on advances in geographic information systems, pp. 99–108

41. Yuan J, Zheng Y, Xie X, Sun G (2011) Driving with knowledge from the physical world. In: Proceedings of the 17th ACM SIGKDD International conference on knowledge discovery and data mining, pp. 316–324
42. Yue M, Kang C, Andris C, Qin K, Liu Y, Meng Q (2018) Understanding the interplay between bus, metro, and cab ridership dynamics in shenzhen, china. Trans GIS 22(3):855–871