Chapter 10
Offline Trajectory Analytics

Panagiotis Tampakis, Stylianos Sideridis, Panagiotis Nikitopoulos, Nikos Pelekis, Christos Doulkeridis, and Yannis Theodoridis

Abstract In recent years, there has been observed an "explosion" of trajectory data production due to the proliferation of GPS-enabled devices, such as mobile phones and tablets. This massive-scale data generation has posed new challenges in the data management community in terms of storing, querying, analyzing, and extracting knowledge out of such data. Knowledge discovery out of mobility data is essentially the goal of every mobility data analytics task. Especially in the maritime and aviation domains, this relates to challenging use-case scenarios, such as discovering valuable behavioral patterns of moving objects, identifying different types of activities in a region of interest, environmental fingerprint, etc. In order to be able to support such scenarios, an analyst should be able to apply, at massive scale, several knowledge discovery techniques, such as trajectory clustering, hotspot analysis, and frequent route/network discovery methods.

10.1 Introduction

Location aware devices such as mobile phones, tablets, and automobiles carry numerous networked sensors, which create huge amounts of data that represent some kind of mobility. In addition, the massive participation of individuals on location-based social networks will continue to fuel exponential growth in the production of this kind of data. This enormous volume of data has posed new challenges in the world of mobility data management in terms of storing, querying, analyzing, and extracting knowledge out of them in an efficient way.

P. Tampakis · S. Sideridis · P. Nikitopoulos · N. Pelekis (✉) · C. Doulkeridis · Y. Theodoridis
University of Piraeus, Piraeus, Greece
e-mail: ptampak@unipi.gr; ssider@unipi.gr; nikp@unipi.gr; npelekis@unipi.gr; cdoulk@unipi.gr; ytheod@unipi.gr

G. A. Vouros et al. (eds.), *Big Data Analytics for Time-Critical Mobility Forecasting*, https://doi.org/10.1007/978-3-030-45164-6_10

10.1.1 About Mobility Data Analytics

Concerning the analysis of mobility data, mobility data analytics aim to describe the mobility of objects, to extract valuable knowledge by revealing motion behaviors or patterns, to predict future mobility behaviors or trends, and in general, to generate various perspectives out of data, useful for many other scientific fields. To serve its purpose, mobility data analytics follows a series of steps. Having assured the collection and efficient storage of mobility data, the next step for an analyst is to familiarize with the mobility data by employing a number of techniques (e.g., statistics, data visualization, and visual analytics) to form a compact and complete picture of the available mobility data. Afterwards, the analyst, depending on the application requirements, proceeds to the appropriate preprocessing steps. The goal is to bring mobility data in a form that serves its later usage by various processes and algorithms that respond to given questions. Data preparation is essential for successful mobility data analytics, since low-quality data typically result in incorrect and unreliable conclusions, as mentioned in chapters in Part I. Finally, mobility data are ready for the application of knowledge extraction methods that will satisfy the given application requirements. There are already several analytical methods and algorithms available from the scientific community and an analyst has the capability either to employ some of the existing techniques or implement some ad hoc solutions that better serve the problems' needs.

The overall objective is to develop advanced, beyond current state-of-the-art data analytics methods and tools over the repository of trajectories of moving objects. The challenge to be addressed here is that information is not purely spatiotemporal; it is contextually enhanced by exploiting integrated data. The big data solutions proposed in this chapter focus onto the problems of cluster analysis and motion pattern detection, hotspot analysis, and semantic-aware mobility network discovery.

In more detail, we designed and implemented a scalable distributed trajectory join method, which utilizes the popular MapReduce distributed programming model. This approach plays a key role as it is the building block upon which our clustering analytics methods are based, as it tackles the scalability bottleneck problem present in mobility data. In addition, we devised and implemented a novel, scalable distributed (sub)trajectory clustering method, which utilizes the aforementioned distributed trajectory join method in order to cluster massive-scale datasets of trajectories. The goal of this approach, that is the upshot of our clustering methods, is to provide a hybrid solution for the whole-trajectory as well as for the subtrajectory clustering problems in an efficient and scalable way. Furthermore, we designed and implemented a scalable distributed trajectory-based hotspot analysis method. In this line of research, we followed a different clustering approach that provides statistical guarantees for the identified clusters. More interestingly, with this approach we solve a different clustering problem that is also inherent in the maritime and aviation domains. Specifically, as a proof of concept, with this approach, we were able to discover hotspots that in the maritime domain can be used to measure the fishing pressure at sea, while in the aviation domain it

identifies air-blocks that present demand-capacity problems. Finally, we designed and implemented a method that discovers mobility networks, which consist of synthetic, pattern-based, compact representations of data. This method employs a semantic-aware methodology, applicable for big contextually enhanced trajectory data (actually the synopses generated the respective component), which results in a network representation of mobility data (actually, spatial graphs enhanced with thematic annotations) that can be utilized to support prediction/forecasting problems.

The rest of the chapter is organized as follows: Section 10.2 familiarizes the reader with some background knowledge. In more detail, Sect. 10.2.1 presents sufficient background knowledge about (sub)trajectory clustering, Sect. 10.2.2 about hotspot analysis, and Sect. 10.2.3 about enriched mobility networks. Subsequently, in Sect. 10.3 we present two big data solutions to the problem of (sub)trajectory clustering and more specifically Sect. 10.3.1.1 employs off-the-shelf clustering algorithms provided by Spark MLib in order to identify clusters of entire trajectories, while Sect. 10.3.2 presents a highly scalable solution to the problem of *Distributed Subtrajectory Clustering*. Moreover, Sect. 10.4 introduces the reader with a big data solution to the problem *Distributed Hotspot Analysis* and Sect. 10.5 with the discovery of *Semantic-aware Mobility Networks* in a distributed way. Section 10.6 presents the related state-of-the-art approaches to the problems identified in this chapter, and finally, Sect. 10.7 concludes the chapter.

10.2 Background

10.2.1 (Sub)trajectory Clustering

Given a set D of moving object trajectories, a trajectory $r \in D$ is a sequence of timestamped locations $\{r_1, \ldots, r_N\}$. Each $r_i = (loc_i, t_i)$ represents the i-th sampled point, $i \in 1, \ldots, N$ of trajectory r, where N denotes the length of r (i.e., the number of points it consists of). Moreover, loc_i denotes the spatial location (2D or 3D) and t_i the time coordinate of point r_i, respectively.

A subtrajectory $r_{i,j}$ is a subsequence $\{r_i, \ldots, r_j\}$ of r which represents the movement of the object between t_i and t_j, where $i < j$ and $i, j \in 1, \ldots, N$. Let $d_s(r_i, s_j)$ denote the spatial distance between two points $r_i \in r$, $s_j \in s$. In our case we adopted the Euclidean distance; however, other distance functions might be applied. Also, let $d_t(r_i, s_j)$ denote the temporal distance, defined as $|r_i.t - s_j.t|$. Furthermore, let Δt_r symbolize the duration of trajectory r (similarly for subtrajectories).

10.2.2 Hotspot Analysis

In geospatial analysis, a hotspot is a geographic area that contains unusually high concentration of activities (e.g., moving objects). The difference between a hotspot and a cluster is that the former aims to discover areas that are statistically significant, whereas the latter focuses on grouping similar objects.

Statistical significance determines whether the relationship between two or more variables is caused by chance. For example, the number of vehicles moving in a specific geospatial area is statistically significant, if it can be proved that it is not the result of chance. In statistical hypothesis testing, the statistical significance can be determined by testing the null hypothesis. To this end, a p-value (probability value) needs to be calculated which represents the probability that the relationship between two or more variables rejects the null hypothesis. In geospatial analysis a p-value can be determined by calculating a z-score value for a given geospatial area. Such z-scores can be calculated by using several geospatial statistics defined in literature, namely the Getis–Ord statistic [23] or the Moran's I [18].

Motivated by the need for big data analytics over trajectories of moving objects, we focus on discovering *trajectory hotspots* in the maritime domain, as this relates to various challenging use-case scenarios [5], as, for instance, detecting fishing pressure, as discussed in Part I of this book. Trajectory hotspot analysis is related to geospatial hotspot analysis, since both discover hotspots on geographical areas. However, the former analysis has two main differences: (a) it considers an additional variable, namely the temporal dimension in the z-score calculation, and (b) it discovers hotspots based on trajectories of objects rather than individual traced object locations. In the following we formally define the problem of trajectory hotspot analysis.

10.2.3 Data-Enriched Mobility Networks

In both the maritime and the aviation domain, we notice a plethora of moving objects. At the same time with the advances in tracking technology there is an abundance of information, not always valuable, concerning the movement of such objects. This has enabled a wide spectrum of novel applications and services. Among them is the process of using the traces of moving entities to produce maps of transportation networks.

This quantity of information leads to the need of discovering new efficient and effective ways to infer the underlying transportation network driven by the data of moving objects itself. Although both ships and aircraft should follow predefined movement plans, there exist many cases where, for various reasons (weather, protected areas, congestion, etc.), objects do not follow these routes plans. Such deviations are crucial to be instantly identified so that preventative measures can be taken to avoid the occurrence of safety-compromising events, such as natural disasters or accidents.

10.3 Distributed Trajectory Clustering

As mentioned in Sect. 10.1, one of the challenges when trying to extract knowledge out of mobility data is cluster analysis, which aims at identifying clusters of moving objects, as well as detecting moving objects that demonstrate abnormal behavior and can be considered as outliers. By discovering these clusters, the underlying hidden patterns of collective behavior can be revealed.

10.3.1 Distributed Whole-Trajectory Clustering

The research so far has focused mainly on dealing with the trajectory clustering problem in a centralized way. However, the problem that we are trying to deal with in this section is that of whole-trajectory clustering in a distributed way. The intuition here is to use the off-the-shelf clustering algorithms provided by Spark MLlib, such as k-means, Gaussian mixture model, power iteration clustering, and bisecting k-means, in order to identify clusters of entire trajectories. To achieve this, we need to transform each trajectory into a vector that will be given as input to the respective clustering algorithm of Spark MLlib.

10.3.1.1 Clustering Trajectories in a Distributed Way with Spark MLlib

The solution to the distributed whole-trajectory clustering problem proposed here is pretty simple, since it utilizes off-the-shelf algorithms of Spark MLlib. In order to achieve this, we just need to transform each trajectory into a N-dimensional vector, where N is the number of samples that constitute a trajectory. Subsequently, these vectors will be given as input to the respective clustering algorithm of Spark MLlib. The actual challenge here is how to transform a dataset of trajectories into vectors, which will be the input for a series of clustering algorithms included in Spark MLlib. In order to vectorize each trajectory, some kind of resampling needs to be performed, due to the fact that different trajectories might have different number of samples. Selecting the resampling method is a crucial decision because it determines what kind of clustering (spatial-only or spatiotemporal) we will end up in performing. This applies because, implicitly, the position j inside a vectorized trajectory determines the time of the observation of v_j. In this analysis, two alternative resampling methods are adopted: the interpolation method and the normalized interpolation method.

More specifically, a trajectory vector v is a vector (2D or 3D) representing a trajectory. The size of the vector for each trajectory $T \in D$ has a fixed length s which is defined a priori. Each element v_j of v, where $j \in 1, \ldots, s$, represents the spatial location (2D or 3D) of the respective trajectory. The interpolation method, as depicted in Algorithm 10.1, takes as input a set of trajectories D and the vector length s and outputs a set D' of vectorized trajectories. In more detail, for each trajectory T (line 3) it creates a vector of size s (line 4) by selecting s timestamps st_j, with $j \in 1, \ldots, s$, of duration $(TN.t - T1.t)/s$, where $(TN.t - T1.t)$ is the

duration of trajectory T, starting from $st_1 = T1.t$ and ending at $st_s = TN.t$. For each st_j, the algorithm finds the corresponding position of T by performing cubic interpolation and stores this position in v_j (lines 5–6). Finally, v is added to D' (line 6) and when all trajectories of D get vectorized, D' is returned (line 9).

Algorithm 10.1 Vectorization by interpolation

```
1: Input: D, s
2: Output: D'
3: for each T ∈ D do
4:     create v;
5:     for j = 1 ... s do
6:         st_j = T1.t + (j − 1)(TN.t − T1.t)/s;
7:         v_j ← cubic_interpolation(T, st_j);
8:     v → D';
9: return D';
```

Now, let us consider two trajectories q and r and their vectorized versions $v(q)$ and $v(r)$, respectively, with $q_N.t - q_1.t \neq r_N.t - r_1.t$. This means that for any given $j \in 1, \ldots, s$, $v(q)_j$ will depict the position of trajectory q at a different timestamp than the position depicted for trajectory r by $v(r)_j$. However, in order for the algorithms that will be employed to perform spatiotemporal clustering, for any given pair of trajectories q and r and for any given $j \in 1, \ldots, s$, it must hold that $v(q)_j$ and $v(r)_j$ represent the position of trajectory q and r at the same timestamp. For this reason, the vectorization by interpolation is used to perform spatial-only clustering. However, if the goal is to perform spatiotemporal trajectory clustering the interpolation vectorization method has the aforementioned shortcoming and in order to overcome this, another vectorization method needs to be employed. For this reason, we propose the normalized interpolation vectorization method which takes as input a set of trajectories D, the vector length s, and the time of the temporally first and last observed sample $D.t_i$ and $D.t_e$, respectively, of D and output a set D' of vectorized trajectories.

In more detail, as depicted in Algorithm 10.2, we first create the universal resampling vector rsv (line 3) by selecting s timestamps st_j, with $j \in 1, \ldots, s$, of duration $(D.t_e - D.t_i)/s$, where $(D.t_e - D.t_i)$ is the duration of the dataset, starting from $st_1 = D.t_i$ and ending at $st_s = D.t_e$ (lines 4–6). The utility of rsv is to help resample each $T \in D$ in such a way so that for any given pair of trajectories q and r and for any given $j \in 1, \ldots, s$, it holds that $v(q)_j$ and $v(r)_j$ represent the position of trajectory q and r at the same timestamp. Subsequently, for each trajectory T we create a vector v of size s (lines 7–8). Then, for each sample j, we examine whether the corresponding time in rsv_j is contained by the lifespan of T (lines 9–10). If it is contained, then we find the corresponding position of T by performing cubic interpolation and we store this position in v_j (line 11). If rsv_j is not contained by the lifespan of T and rsv_j is less or equal to the first timestamp of the trajectory $T_1.t$, then the first recorded position of T is stored in v_j (lines 13). Otherwise, if rsv_j is not contained by the lifespan of T and rsv_j is greater or equal to the last

Algorithm 10.2 Vectorization by normalized interpolation

```
1: Input: D, s, D.t_i and D.t_e
2: Output: D'
3: create rsv;
4: for j = 1 ... s do
5:     st_j = D.t_i+(j − 1)(D.t_e − D.t_i)/s;
6:     rsv_j ← st_j;
7: for each T ∈ D do
8:     create v;
9:     for j = 1 ... s do
10:        if rsv_j ∈(T_1.t, T_N.t) then
11:            v_j ← cubic_interpolation(T, rsv_j);
12:        else if rsv_j ≤ T_1.t then
13:            v_j ← T_1.p
14:        else
15:            v_j ← T_N.p
16:    v → D';
17: return D';
```

timestamp of the trajectory $T_N.t$, then the last recorded position of T is stored in v_j (line 15). Finally, v is added to D' (line 16) and when all trajectories of D get vectorized, D' is returned (line 17).

10.3.2 Distributed Subtrajectory Clustering

However, identifying clusters of complete trajectories can result in disregarding significant patterns that might exist only for some portions of their lifespan. The following motivating example shows the merits of subtrajectory clustering.

Example 10.1 (Subtrajectory Clustering) Figure 10.1a illustrates six trajectories moving in the xy-plane, where each one of them has a different origin–destination pair. More specifically, these pairs are $A \rightarrow B$, $A \rightarrow C$, $A \rightarrow D$, $B \rightarrow A$, $B \rightarrow C$, and $B \rightarrow D$. These six trajectories have the same starting time and similar speed. A typical trajectory clustering technique would fail to identify any clusters. However, the goal of a subtrajectory clustering method is to identify 4 clusters ($A \rightarrow O$ (red), $B \rightarrow O$ (blue), $O \rightarrow C$ (purple), $O \rightarrow D$ (orange)) and 2 outliers ($O \rightarrow A$ and $O \rightarrow B$ (black)), as depicted in Fig. 10.1b.

The problem of subtrajectory clustering is shown to be NP-Hard (cf. [1]). In addition, the objects to be clustered are not known beforehand (as in entire-trajectory—from now on—clustering algorithms), but have to be identified through a trajectory segmentation procedure. Efforts that try to deal with this problem in a centralized way do exist [1, 15, 28]; however, applying these centralized algorithms over massive data in a scalable way is far from straightforward. This calls for parallel and distributed algorithms that address the scalability requirements.

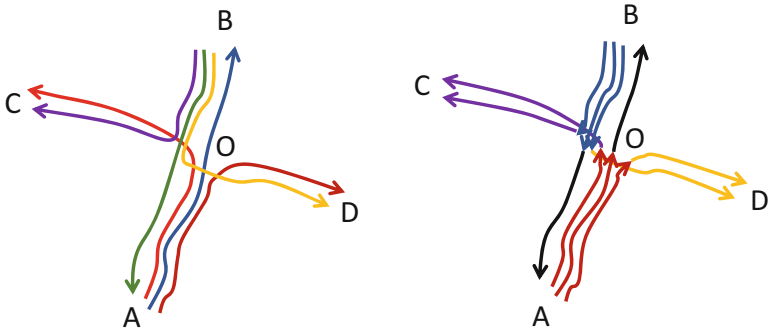


Fig. 10.1 (**a**) Six trajectories moving in the xy-plane and (**b**) 4 clusters (red, blue, orange, and purple) and 2 outliers (black)

Algorithm 10.3 $DSC(D)$

Input: D
Output: set C of clusters, set O of outliers
Preprocessing: *Repartition* D;
for each partition $D_i \in \cup_{i=1}^{P} D_i$ **do**
    perform *Point-level Join*;
group by *Trajectory*;
for each Trajectory $r \in D$ **do**
    perform *Subtrajectory Join*
    perform *Trajectory Segmentation*;
group by D_i;
for each subtrajectory $r' \in D_i$ **do**
    calculate $Sim(r', s')\ \forall s' \in D_i$;
perform *Clustering*;
perform *Refine Results*;
return C and O;

10.3.2.1 Definitions

Subtrajectory clustering relies on the use of a similarity function between subtrajectories. Although various similarity measures have been defined in the literature, our choice of similarity function is motivated by the following (desired) requirements: **variable sampling rate and lack of alignment**, **variable trajectory length**, **temporal displacement**, **symmetry**, and **efficiency**.

In order to meet with the aforementioned specifications we utilize the longest common subsequence (LCSS) for trajectories, as defined in [36].

$$Sim(r, s) = \frac{LCSS_{\epsilon_t, \epsilon_{sp}}(r, s)}{min(|r|, |s|)} \tag{10.1}$$

where $|r|$ ($|s|$) is the length of r (s, respectively). Moreover, it holds that $Sim(r, s) = Sim(s, r)$.

However, LCSS considers as equally similar all the points that exist within an ϵ_{sp} spatial range from r, which is a fact that might compromise the quality of the clustering results. Ideally, given two matching points $r_i \in r$ and $s_j \in s$, s_j (r_i, respectively) should contribute to $LCSS_{\epsilon_t,\epsilon_{sp}}(r, s)$, proportionally to the distance $d_s(r_i, s_j)$. For this reason, we propose a "weighted" LCSS similarity between trajectories, that incorporates the aforementioned distance proportionality. In more detail:

$$Sim(r, s) = \frac{\Sigma_{k=1}^{min(|r|,|s|)}\left(1 - \frac{d_s(r_k,s_k)}{\epsilon_{sp}}\right)}{min(|r|, |s|)} \tag{10.2}$$

where (r_k, s_k) is a pair of matched points.

Our approach to subtrajectory clustering splits the problem into three steps. The first step is to retrieve for each trajectory $r \in D$ all the moving objects, with their respective portion of movement, that moved close enough in space and time with r, for at least some time duration. This is a well-defined problem in the literature of mobility data management, known as *subtrajectory join* (the case of self-join). The subtrajectory join will return for each pair of (sub)trajectories, all the common subsequences that have at least some time duration, which are actually candidates for the longest common subsequence. Formally:

Problem 10.1 (Subtrajectory Join) Given a temporal tolerance ϵ_t, a spatial threshold ϵ_{sp}, and a time duration δt, retrieve all pairs of subtrajectories $(r', s') \in D$ such that: (a) for each pair $\Delta t_{r'}, \Delta t_{s'} \geq \delta t$, (b) $\forall r_i \in r'$ there exists at least one $s_j \in s'$ so that $d_s(r_i, s_j) \leq \epsilon_{sp}$ and $d_t(r_i, s_j) \leq \epsilon_t$, and (c) $\forall s_j \in s'$ there exist at least one $r_i \in r'$ so that $d_s(s_j, r_i) \leq \epsilon_{sp}$ and $d_t(s_j, r_i) \leq \epsilon_t$.

The second step takes as input the result of the first step and aims at segmenting each $r \in D$ into a set of subtrajectories. In our case, the way that a trajectory is segmented into subtrajectories is neighborhood-aware, meaning that a trajectory will be segmented every time its neighborhood changes significantly.

Problem 10.2 (Trajectory Segmentation) Given a trajectory r, discover the set of timestamps CP (cutting points), where the neighborhood (the density or the composition) of r changes significantly. Then according to CP, r is partitioned to a set of subtrajectories $\{r'_1, \ldots, r'_M\}$, where $M = |CP| + 1$ is the number of subtrajectories for a given trajectory r, such that $r = \cup_{k=1}^{M} r'_k$ and $k \in [1, M]$.

Given the output of Problem 10.1, applying a trajectory segmentation algorithm for the trajectories D will result in a new set of subtrajectories D'. The third step takes as input D' and the goal is to create clusters (whose cardinality is unknown) of similar subtrajectories and at the same time identify subtrajectories that are significantly dissimilar from the others (outliers). More specifically, let $C = \{C_1, \ldots, C_K\}$ denote the clustering, where K is the number of clusters, and for

every pair of clusters C_i and C_j, with $i, j \in [1, K]$, it holds that $C_i \cap C_j = \emptyset$. Now, let us assume that each cluster $C_i \in C$ is represented by one subtrajectory $R_i \in C_i$, called *Representative*. Furthermore, let R denote the set of all representatives. Actually, the problem of clustering is to discover clusters of objects such that the intra-cluster similarity is maximized and the inter-cluster similarity is minimized. Therefore, if we ensure that the similarity between the representatives is zero, then the problem of subtrajectory clustering can be formulated as an optimization problem as follows.

Problem 10.3 (Subtrajectory Clustering and Outlier Detection) Given a set of subtrajectories D', partition D' into a set of clusters C, and a set of outliers O, where $D' = C \cup O$, in such a way so that the sum of similarity between cluster members and cluster representatives (*SSCR*) is maximized:

$$SSCR = \Sigma_{\forall R_i \in R} \Sigma_{\forall r'_j \in C_i} Sim(R_i, r'_j) \tag{10.3}$$

However, trying to solve Problem 10.3 by maximizing Eq. (10.3) is not trivial, since the problem to segment trajectories to subtrajectories, select the set of representatives R and its cardinality $|R|$ that maximizes Eq. (10.3), has combinatorial complexity.

Here, we address the challenging problem of subtrajectory clustering in a distributed setting, where the dataset D is distributed across different nodes, and centralized processing is prohibitively expensive.

Problem 10.4 (Distributed Subtrajectory Clustering) Given a distributed set of trajectories, $D = \cup_{i=1}^{P} D_i$, where P is the number of partitions of D, perform the subtrajectory clustering task in a parallel manner.

Actually, Problem 10.4 can be broken down to solving Problems 10.1–10.3 (in that order) in a parallel/distributed way. In what follows, we adopt this approach and outline a solution that is based on MapReduce.

The overall subtrajectory clustering algorithms are presented in Algorithm 10.3. Each of the major steps in this algorithm is presented in subsequent paragraphs. Initially, we *Repartition* the data into P equi-sized, temporally sorted temporal partitions (files), which are going to be used as input to the distributed subtrajectory join algorithm (line 3). Note that this is actually a preprocessing step that only needs to take place once for each dataset D. However, it is essential as it enables load balancing by addressing the issue of temporal skewness in the input data. Subsequently, for each partition $D_i \in \cup_{i=1}^{P} D_i$ and for each trajectory (the reference trajectory) we discover parts of other trajectories that move close enough in space and time (line 5). Successively, we group by reference trajectory in order to perform the subtrajectory join (line 8). At this phase, since our data is already grouped by trajectory, we also perform trajectory segmentation in order to split each trajectory to subtrajectories (line 9). In turn, we utilize the temporal partitions created during the *Repartition* phase and re-group the data by temporal partition. For each $D_i \in \cup_{i=1}^{P} D_i$ we calculate the similarity between subtrajectories and perform the clustering procedure (line 12). At this point we should mention that

if a subtrajectory intersects the borders of multiple partitions, then it is replicated in all of them. This will result in having duplicate and possibly contradicting results. For this reason, as a final step, we specify the *Refine Results* procedure (line 14). Finally, a set C of clusters and a set O of outliers are produced.

10.3.2.2 Distributed Subtrajectory Join

As already mentioned, the first step is to perform the subtrajectory join in a distributed way. For this reason, we exploit the work presented in [34], coined *DTJ* (*D*istributed sub*T*rajectory *J*oin), which introduces an efficient and highly scalable approach to deal with the subtrajectory join problem (Problem 10.1) by means of MapReduce. More specifically, *DTJ* is comprised of a *Repartitioning* phase and a *Query* phase. The *Repartitioning* phase is a preprocessing step that takes place only once and it is independent of the actual parameters of the problem, namely ϵ_{sp}, ϵ_t, and δt. The goal for this step is to produce load balanced temporal partitions. The idea is to construct an equi-depth histogram based on the temporal dimension, where each of the M bins contains the same number of points and the borders of each bin correspond to a temporal interval $[t_i, t_j)$. The histogram is constructed by taking a sample of the input data. Then, the input data is partitioned to processing tasks based on the temporal intervals of the histogram bins. This guarantees temporal locality in each partition, as well as equi-sized partitions, thus balancing the load fairly.

In the *Query* phase, the actual join processing takes place. It consists of two steps, the *Join* and the *Refine* step, which are implemented as a *Map* and a *Reduce* function, respectively. The output of this MapReduce job is, for each reference trajectory $r \in D$, all the moving objects, with their respective portion of movement, that moved close enough with r in space and time for at least some time duration. In Fig. 10.2, the *DTJ* query corresponds to Job 1 until the Refine() procedure.

For more technical details about the algorithms involved in *DTJ*, their complexity and an extensive experimental study, we refer to [34].

10.3.2.3 Distributed Trajectory Segmentation

The *Trajectory Segmentation* Algorithm (TSA) takes as input a single trajectory, along with information about its neighborhood, and partitions it to a set of subtrajectories. We propose two alternative segmentation algorithms. The first algorithm, coined TSA_1, identifies the beginning of a new subtrajectory whenever the density of its neighborhood changes significantly. For this purpose, we use the concept of *voting* as a measure of density of the surrounding area of a trajectory. For a given point r_i and any trajectory s, the voting $V(r_i)$ is defined as:

$$V(r_i) = \Sigma_{\forall s \in D} \frac{d_s(r_i, s_k)}{\epsilon_{sp}} \tag{10.4}$$

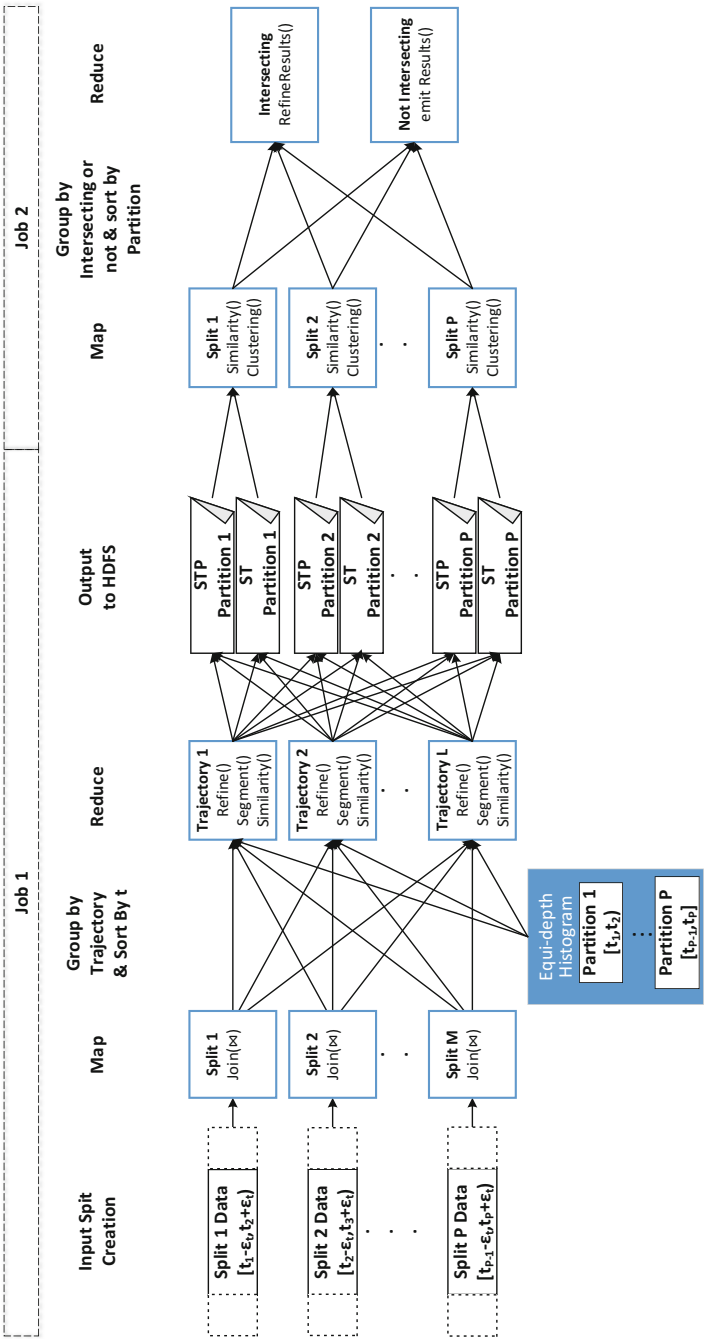


Fig. 10.2 The *DSC* algorithm. (Job 1) *DTJ* and *Trajectory Segmentation* and (Job 2) *Clustering* and *Refine Results*

where s_k is the matching point of s with r_i, as emitted by the subtrajectory join procedure.

Finally, the voting of a trajectory (or subtrajectory) is defined as:

$$V(r) = \frac{1}{N} \Sigma_{i=1}^{N} V(r_i) \tag{10.5}$$

The second segmentation algorithm, coined TSA_2, identifies the beginning of a new subtrajectory whenever the composition of its neighborhood changes substantially. This algorithm takes as input a list $L(r_i)[]$ of the trajectory ids that have been produced as output by the *DTJ* procedure. The following example explains intuitively the difference between the two segmentation algorithms.

Example 10.2 Consider the example of Fig. 10.3a that illustrates five trajectories: $A \rightarrow B$, $A \rightarrow C$, $A \rightarrow D$, $C \rightarrow B$, and $D \rightarrow B$. Figure 10.3b and c depict the result of TSA_1 and TSA_2, respectively. In more detail, we can observe that both TSA_1 and TSA_2 segmented trajectory $A \rightarrow D$ to subtrajectories $A \rightarrow O$ and $O \rightarrow D$, due to the fact that after O, both the density and the composition of the neighborhood change. The same holds for trajectories $A \rightarrow C$, $C \rightarrow B$, and $D \rightarrow B$, which are segmented to subtrajectories $A \rightarrow O$, $O \rightarrow C$, $C \rightarrow O$, $O \rightarrow B$, $D \rightarrow O$, and $O \rightarrow B$. However, when it comes to trajectory $A \rightarrow B$, we can observe that while TSA_2 segments it to subtrajectories $A \rightarrow O$ and $O \rightarrow B$, TSA_1 does not perform any segmentation. This is due to the fact that, after O, even though the density of the neighborhood remains the same (i.e., 3 moving objects), the composition of the neighborhood changes completely.

Both segmentation algorithms share a common methodology, which employs two consecutive sliding windows W_1 and W_2 of size w (i.e., w samples) to estimate the point $r_i \in CP$ (cutting point) where the "difference" between the two windows is maximized. This methodology has been successfully applied in the past on signal segmentation [24, 25]. To exemplify, let us consider trajectory $A \rightarrow D$ of Example 10.2. For simplicity, we assume that the voting of the specific trajectory from A to O is 3 and from O to D is 1. Figure 10.4 illustrates the two sliding windows W_1 and W_2 that traverse the voting signal of trajectory $A \rightarrow D$.

Similar Subtrajectories The next step is to calculate the similarity between all the pairs of subtrajectories, using Eq. (10.2). This cannot be done completely after the segmentation at the *Reducer* phase of Job 1, illustrated in Fig. 10.2, because at that point each reduce function has information only about the segmentation of the reference trajectory to subtrajectories. For this reason, at this point we cannot calculate the denominator of Eq. (10.2). However, for each subtrajectory $r' \in r$, where r is the reference trajectory, we can calculate the similarity between the matching points (enumerator of Eq. (10.2)).

At this point, each *Mapper* has now all the information needed to calculate the similarity between all the pairs of subtrajectories (Eq. (10.2)), for each temporal partition separately. The similarity between subtrajectories is output in a new relation,

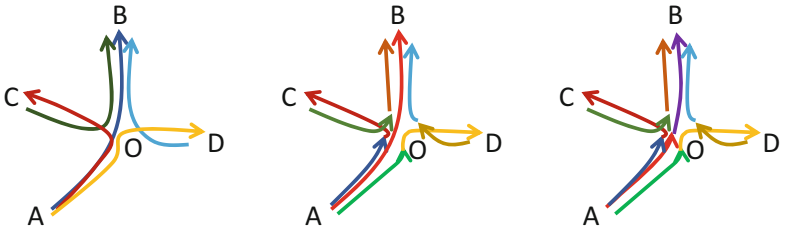


Fig. 10.3 (**a**) Five trajectories $A \rightarrow B$, $A \rightarrow C$, $A \rightarrow D$, $C \rightarrow B$, and $D \rightarrow B$, (**b**) TSA_1 segmentation, (**c**) TSA_2 segmentation

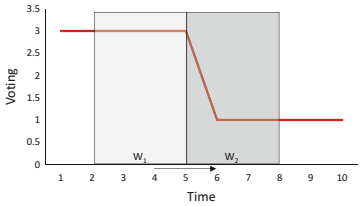


Fig. 10.4 The two consecutive sliding windows W_1 and W_2 used by the segmentation algorithms

called *SP*. Each tuple of this relation holds information about a subtrajectory r' and its similarity with all the other subtrajectories, whenever this similarity is larger than zero. More specifically, *SP* contains a set of key-value pairs where the key is the ID of the subtrajectory ($r'.ID$) and the value is a list $AdjLst$ containing elements of the form ($s'.ID$, Sim), where s' is a subtrajectory for which it holds that $Sim(r', s') > 0$.

10.3.2.4 Distributed Subtrajectory Clustering

Clustering After having calculated the similarity between all pairs of subtrajectories for each temporal partition, we can proceed to the actual clustering and outlier detection procedure. The intuition behind the proposed solution to the subtrajectory clustering and outlier detection problem (Problem 10.3) is to select as cluster representatives highly voted subtrajectories that have zero similarity with the already selected representatives $R_i \in R$, thus addressing the inter-cluster distance minimization. Then, we assign each subtrajectory to the cluster (i.e., Representative) with which it has the greatest similarity.

The input of the clustering algorithm is SP, ST, and parameters k and α and the output is the set of clusters C and the set of outliers O. More specifically, k is a threshold for setting a lower bound on the voting of a representative. This prevents the algorithm from identifying clusters with small support. Parameter α is a similarity threshold used to assign subtrajectories to cluster representatives.

It ensures that a subtrajectory assigned to a cluster has sufficient similarity with the representative of the cluster. This actually poses a lower bound to the average distance between the representatives and the cluster members and, consequently, guarantees a minimum quality in the identified clusters (intra-cluster distance).

Refinement of Results At this point we successfully accomplished to deal with Problem 10.3 for each temporal partition. However, this might result in having duplicates due to the fact that each subtrajectory that temporally intersects multiple partitions is replicated to each one of them. The actual problem that lies here is not the duplicate elimination problem itself but the fact that the result for such a subtrajectory might be contradicting in different partitions. In more detail, for each partition, the clustering procedure will decide whether a subtrajectory is a *Representative* (R), a *Cluster Member* (C), or an *Outlier* (O). Hence, for each intersecting subtrajectory q and for each pair of consecutive temporal partitions (i, j) with which q intersects, q can have the following pairs of states: (a) O–O, (b) R–R, (c) C–C, (d) R–C (C–R), (e) R–O (O–R), and (f) C–O (O–C).

For each of the above cases a decision has to be made, in order to eliminate duplicates and provide the correct result according to the problem definition. More specifically, in case of (a), q is marked as outlier in both partitions; hence, we only need to eliminate duplicates. In case of (b), the two clusters are "merged," since all of the subtrajectories that belong to them are similar "enough" with q, which is the representative of both clusters. In case of (c), let us assume that q belongs to cluster $C_i(R(q))$ in Partition i and $C_{i+1}(R(q))$ in Partition $i + 1$. Then, q is assigned to the cluster with which it has the largest similarity with its representative. In case of (d), q remains to be a cluster representative and is removed from the cluster C in which it is a member. Finally, in case of (e) and (f), q is removed from O. For more details concerning the *Distributed Subtrajectory Clustering* solution presented in this section, please refer to [35].

10.3.2.5 Experimental Results

In this section, we provide our experimental study on the solution that we proposed in order to address the *Distributed Subtrajectory Clustering* problem. The datasets employed for our experiments are:

- **IFS (April 2016)**—Flights between Madrid and Barcelona during April 2016 of size 43 MB and consisting of approximately 900K records.
- **NARI/Brest Area (6 months) Raw**—Vessels moving in Brest area, consisting of approximately 18 million records of size 697 MB.
- **FlightAware (April 2016)**—Trajectories of aircraft that consist of approximately 250 million records of 11.4 GB.
- **IMIS (3 years)**—consists of 699,031 trajectories of ships moving in the Eastern Mediterranean for a period of 3 years. This dataset contains approximately 1.5 billion records, 56 GB in total size.

Our experimental methodology is as follows: Initially, we illustrate that the subtrajectory clustering solution produces results of high quality as compared to the whole-trajectory clustering solution. Finally, we verify the scalability of our algorithms by varying the dataset size.

The experiments were conducted in a 49 node Hadoop 2.7.2 cluster. One node acted as the master and 48 nodes acted as slaves. The master node consists of 8 CPU cores, 8 GB of RAM, and 60 GB of hard disk, while each slave node is comprised of 4 CPU cores, 4 GB of RAM, and 60 GB of hard disk. Our configuration enables each slave node to launch 4 containers, thus resulting that at a given time the cluster can run up to 192 jobs (Map or Reduce). The operating system running on all the nodes is Debian 8.3.

Quality of Clustering Analysis To illustrate the quality of the results we employed the **IFS (April 2016)** (Fig. 10.5a). To assist with our analysis, we performed a preprocessing step where all trajectories were aligned to start at the same time. In order to be able to compare the two approaches (i.e., whole- and subtrajectory clustering), we deactivated the segmentation step of the subtrajectory clustering solution. The subtrajectory clustering algorithm identified 6 clusters, 3 clusters from Madrid to Barcelona and 3 clusters from Barcelona to Madrid. Moreover, an outlier was detected, which is not something common in aviation data. Both the cluster representatives and the outlier are depicted in Fig. 10.5b. However, if we consider only the spatial dimension, these 6 clusters and 1 outlier seem to be actually 2 clusters. But if we also take into consideration the temporal dimension, as presented in the space-time cube of Fig. 10.5d, we will actually see that there are clusters that follow the same path but have different behavior as far as it concerns the speed and/or the duration of the flight. These probably correspond to different types of aircrafts. In order to compare with the Spark MLlib-based solution, we vectorized the data and utilized the k-means algorithm on the same dataset with $k = 6$, which is the number of clusters that was previously identified. Figure 10.5c illustrates the result of k-means and Fig. 10.5d the corresponding space-time cube.

To conclude, the distributed subtrajectory clustering approach presents several advantages over the Spark MLlib-based. To begin with, an important issue is that the number of clusters is discovered by the algorithm and is not up to the user to give as input the correct number of clusters. In addition, another important functionality that a trajectory clustering algorithm should have is the outlier detection. Finally, due to the fact that the Spark MLlib-based solution with both vectorization methods does some kind of resampling, some movement patterns might be "lost," depending on the number of out samples of the resampling procedure.

Scalability We vary the size of our dataset and measure the execution time of our algorithms. To study the effect of dataset size, we created 4 datasets: 20%, 40%, 60%, and 80% of each of the original datasets. For the purpose of this analysis

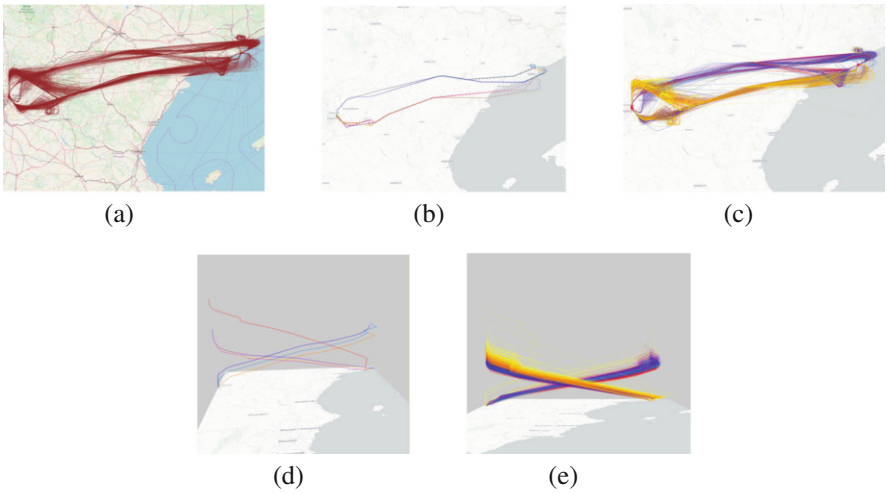


Fig. 10.5 (**a**) Raw data, (**b**) cluster representatives (6 clusters discovered), (**c**) k-means with $k = 6$, (**d**) cluster representatives—space-time cube, (**e**) k-means with $k = 6$—space-time cube

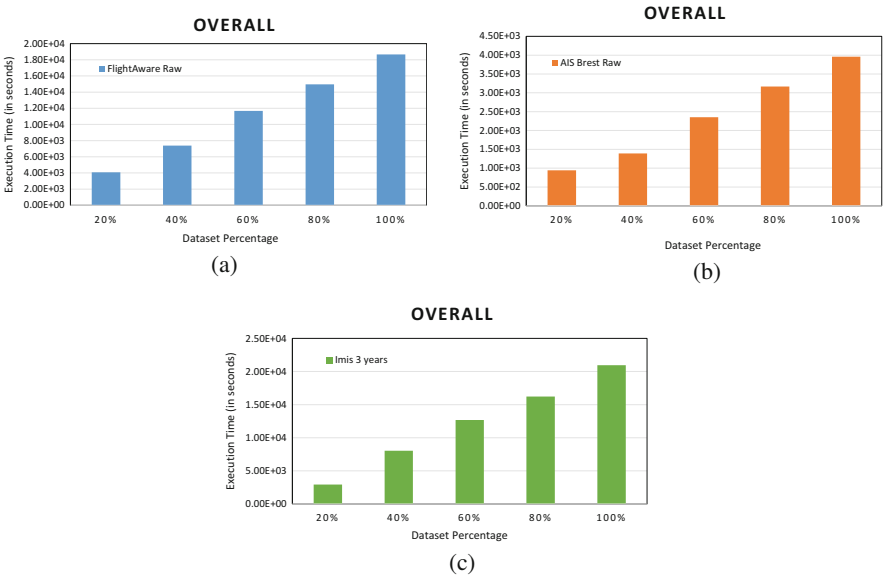


Fig. 10.6 Scalability analysis varying the size of the (**a**) FlightAware (April 2016), (**b**) NARI/Brest Area (6 months) Raw, and (**c**) IMIS (3 years) dataset

we are going to use the following datasets: **NARI/Brest Area (6 months) Raw**, **FlightAware (April 2016)**, and **IMIS (3 years)**.

As shown in Fig. 10.6, as the size of the dataset increases, the proposed clustering algorithm turns out to have linear behavior.

10.4 Distributed Hotspot Analysis

In this section, we present the THS (trajectory hotspot) and aTHS (approximate trajectory hotspot) algorithms for distributed hotspot analysis over big trajectory data, as presented in [20]. Our approach to Hotspot discovery and analysis is based on spatiotemporal partitioning of the 3D data space in cells. Accordingly, we try to identify cells that constitute hotspots, i.e., not only do they have high density, but also that the density values are statistically significant. We employ the Getis–Ord statistic [23], a popular metric for hotspot analysis, which produces z-scores and p-values. A cell is considered as a hotspot, if it is associated with high z-score and low p-value. Unfortunately, the Getis–Ord statistic is typically applicable in the case of 2D spatial data, and even though it can be extended to the 3D case, it has been designed for point data. In contrast, our application scenario concerns trajectories of moving objects, temporally sorted sequences of spatiotemporal positions, and the applicability of hotspot analysis based on a metric, such as the Getis–Ord statistic (but also any other metric), is far from straightforward. To this end, we formulate the problem of trajectory hotspot analysis, where our main intuition is that the contribution of a moving object to a cell's density is proportional to the time spent by the moving object in the cell. In particular, we adapt the Getis–Ord statistic in order to capture this intuition for the case of trajectory data. Then, we propose a parallel and scalable processing algorithm for computing hotspots in terms of spatiotemporal cells produced by grid-based partitioning of the data space under study. Our algorithm achieves scalability by parallel processing of z-scores for the different cells and returns the exact result set. Moreover, we couple our exact algorithm with a simple approximate algorithm that only considers neighboring cells at distance h (in number of cells), instead of all cells, thus achieving significant performance improvements. More importantly, we show how to quantify the error in z-score computation, thereby developing a method that can trade-off accuracy for performance in a controlled manner.

10.4.1 Definitions

Consider a database that contains trajectories of moving objects. A trajectory is a sequence of data points p described by 2D geospatial coordinates ($p.x$ for longitude and $p.y$ for latitude), a timestamp ($p.t$), and the moving object's id ($p.o$). Furthermore, consider a spatiotemporal partitioning $\mathscr{P}$ which partitions the 3D spatiotemporal domain into n 3D cells $\{c_1, \ldots, c_n\} \in \mathscr{P}$. Every data point p is mapped to one cell c_i, which is determined based on the spatiotemporal location of p. Individual objects moving inside single cells constitute a subset of data points p which form individual subtrajectories τ. The earliest (latest) point of a subtrajectory τ is denoted as $\tau.p_{start}$ ($\tau.p_{end}$). Also, we use $c_{i_{start}}$, $c_{i_{end}}$ to refer to temporal start and end of a cell c_i.

We also define the *attribute value* x_i of the cell c_i as: $x_i = \Sigma_{\tau \in c_i} \frac{\tau.p_{end}.t - \tau.p_{start}.t}{c_{i_{end}} - c_{i_{start}}}$. Thus every object that moves in a spatiotemporal cell c_i contributes to the cell's attribute value by its temporal duration $\tau.p_{end}.t - \tau.p_{start}.t$ normalized by dividing with the cell's temporal lifespan $c_{i_{end}} - c_{i_{start}}$. This definition implies that the longer a moving object's subtrajectory stays in a spatiotemporal cell, the higher its contribution to the cell's attribute value.

Having calculated an attribute value for each cell, we opt to use the Getis–Ord statistic to calculate z-scores. The Getis–Ord statistic G_i^* is defined as [23]:

$$G_i^* = \frac{\Sigma_{j=1}^n w_{i,j} x_j - \overline{X} \Sigma_{j=1}^n w_{i,j}}{S \sqrt{\frac{[n \Sigma_{j=1}^n w_{i,j}^2 - (\Sigma_{j=1}^n w_{i,j})^2]}{n-1}}} \tag{10.6}$$

where x_j is the attribute value for cell j, $w_{i,j}$ is the spatiotemporal weight between cell i and j, n is equal to the total number of cells, and

$$\overline{X} = \frac{\Sigma_{j=1}^n x_j}{n} \tag{10.7}$$

$$S = \sqrt{\frac{\Sigma_{j=1}^n x_j^2}{n} - (\overline{X})^2} \tag{10.8}$$

The spatiotemporal weight $w_{i,j}$ used in the Getis–Ord statistic represents the score influence between neighboring cells; a cell needs to have a neighborhood of high attribute values to be considered as hotspot. Our goal is to have the influence of a neighboring cell c_i to a given cell c_j to be decreasing with increased spatiotemporal distance. Thus we employ a weight function that decreases exponentially with increasing distance; we define: $w_{i,j} = a^{1-\rho}$, where $a > 1$ is an application-dependent parameter, and ρ represents the distance between cell i and cell j measured in number of cells. For immediate neighboring cells, where $\rho = 1$, we have $w_{i,j} = 1$, while for the next neighbors we have, respectively: $1/a$, $1/a^2$, ...

Based on this, the problem of trajectory hotspot analysis is to identify the k most statistically significant cells according to the Getis–Ord statistic and can be formally stated as follows:

Problem 10.5 (Trajectory Hotspot Analysis) Given a trajectory dataset and a space partitioning $\mathscr{P}$, find the top-k cells $TOPK = \{c_1, \ldots, c_k\} \in \mathscr{P}$ based on the Getis–Ord statistic G_i^*, such that: $G_i^* \geq G_j^*, \forall c_i \in TOPK, c_j \in \mathscr{P} - TOPK$.

Our aim is to study the problem of trajectory hotspot analysis over massive spatiotemporal data by proposing a parallel and scalable solution. Thus, we assume that the trajectory dataset is stored in multiple nodes, without any more specific

assumptions about the exact partitioning mechanism. Hence, here, we study a distributed version of the trajectory hotspot analysis problem.

10.4.2 Exact THS Algorithm

The proposed Exact THS algorithm is designed to be efficiently executed over a set of nodes in parallel and is implemented in Apache Spark. The input dataset D is assumed to be stored in a distributed file system, in particular HDFS. Intuitively, our solution consists of three main steps, which are depicted in Fig. 10.7. In the first step, the goal is to compute all the cells' attribute values of a user-defined spatiotemporal equi-width grid. To this end, the individual attribute values of trajectory data points are aggregated into cell attribute values, using Eq. (10.6). Then, during the second step, we calculate the cells' attribute mean value and standard deviation which will be provided to the Getis–Ord formula later. Furthermore, we compute the weighted sum of the values for each cell $c_i : \Sigma_{j=1}^{n} w_{i,j} x_j$. Upon successful completion of the second step, we have calculated all the individual variables included in the Getis–Ord formula, and we are now ready to commence the final step. The goal of the third step is to calculate the z-scores of the spatiotemporal grid cells by applying the Getis–Ord formula. The trajectory hotspots can then be trivially calculated by selecting either the top-k cells with the higher z-score values or the cells having a p-value below a specified threshold.

10.4.3 An Approximate Algorithm: aTHS

The afore-described algorithm (*THS*) is exact and computes the correct hotspots over widely distributed data. However, its computational cost is relatively high and can be intolerable when the number of cells n in $\mathscr{P}$ is large. This is because every cell attribute value must be sent to all other cells of the grid, thus leading to data exchange through the network of $O(n^2)$ as well as analogous computational cost, which is prohibitive for large values of n. Instead, in this section, we propose an approximate algorithm, denoted *aTHS*, for solving the problem. The rationale behind *aTHS* algorithm is to compute an approximation $\hat{G}_i^*$ of the value G_i^* of a cell c_i by taking into account only those cells at maximum distance h from c_i. The distance is measured in a number of cells. Intuitively, cells that are located far away from c_i will only have a small effect on the value G_i^* and should not affect its accuracy significantly when neglected. More interestingly, we show how to quantify the error $\Delta G_i^* = G_i^* - \hat{G}_i^*$ of the computed hotspot z-score of any cell c_i, when taking into account only neighboring cells at distance h. In turn, this yields an analytical method that can be used to trade-off accuracy with computational efficiency, having bounding error values.

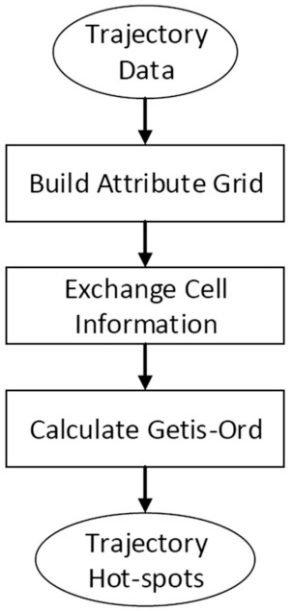


Fig. 10.7 Overview of THS algorithm

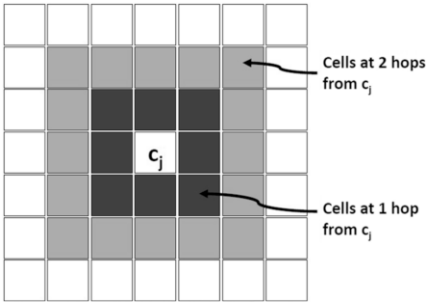


Fig. 10.8 Example of cells at distance from a reference cell c_j (the dark color indicates the weight of their contribution to c_j's value x_j)

The *aTHS* Algorithm Based on the problem definition, cells located far away from a reference cell c_i only have a limited contribution to the Getis–Ord value G_i^* of c_i. Our approximate algorithm (*aTHS*) exploits this concept and can be parameterized with a value h, which defines the subset of neighboring cells that contribute to the value of c_i. We express h in terms of cells, for instance, setting h=2 corresponds to the case depicted in Fig. 10.8, where only the colored cells will be taken into account by *aTHS* for the computation of $\hat{G}_i^*$ (an approximation of the value of G_i^*). In practice, the relationship between cell c_j and white cells can be expressed by

setting their weight factor equal to zero. In algorithmic terms, *aTHS* is differentiated from *THS* in the second and third step.

10.4.4 Experimental Study

In this section, we evaluate the performance of our approach for trajectory hotspot analysis. We implemented all algorithms in Java, using Apache Spark 2.2.0 Core API.

Datasets We employed a real dataset containing surveillance information from the maritime domain. The data was collected over a period of 3 years, consisting of 83,735,633 individual trajectories for vessels moving in the Eastern Mediterranean Sea. This dataset is 89.4 GB in total size and contains approximately 1.9 billion records. Each record represents a point in the trajectory of a vessel and is made up by <*trajectoryID;timestamp;latitude;longitude*>. The dataset is stored in 720 HDFS blocks, in uncompressed text format.

Evaluation Methodology We picked four parameters to study their effect on the efficiency of our algorithm, namely (a) the spatial size of cells (in terms of both latitude and longitude), (b) the temporal size of cells, (c) the h distance which defines the number of neighboring cells contributing to the score of a reference cell c_i, and (d) the k number of hotspots to be reported in the final result set. In practice, the first two parameters affect the number n of cells of P, the third parameter h refers to the number of broadcasting messages which occur during the second step, and the fourth parameter k affects exclusively the last step of our algorithm. Also, we set a equal to 2 in all experiments.

Metrics Our main evaluation metric was the execution time needed for each individual step of our algorithm on the Spark cluster. In the following, the actual execution times will be presented, omitting any overhead caused by Spark and YARN initialization procedures. All execution times are depicted using the number of milliseconds elapsed for processing each step.

We deployed our code on a Hadoop YARN 2.7 cluster consisting of 10 computing nodes. Node 1 has 6 GB of RAM and 4 single-core CPUs running at 2.1 GHz. Nodes 2–10 have 8 GB of RAM, 4 single-core CPUs running at 2.1 GHz, and 100 GB hard disk each. Node 1 is configured to run the HDFS NameNode and YARN ResourceManager services, whereas all other nodes run the HDFS DataNode and YARN NodeManager services. In all our experiments, we use the YARN cluster deploy mode. We initiate 9 Spark executors, configured to use 5.5 GB of main memory and 2 executor cores each. We also configured HDFS with 128 MB block size and a replication factor of 2. On each node, Java 8 is installed on Ubuntu 16.04.

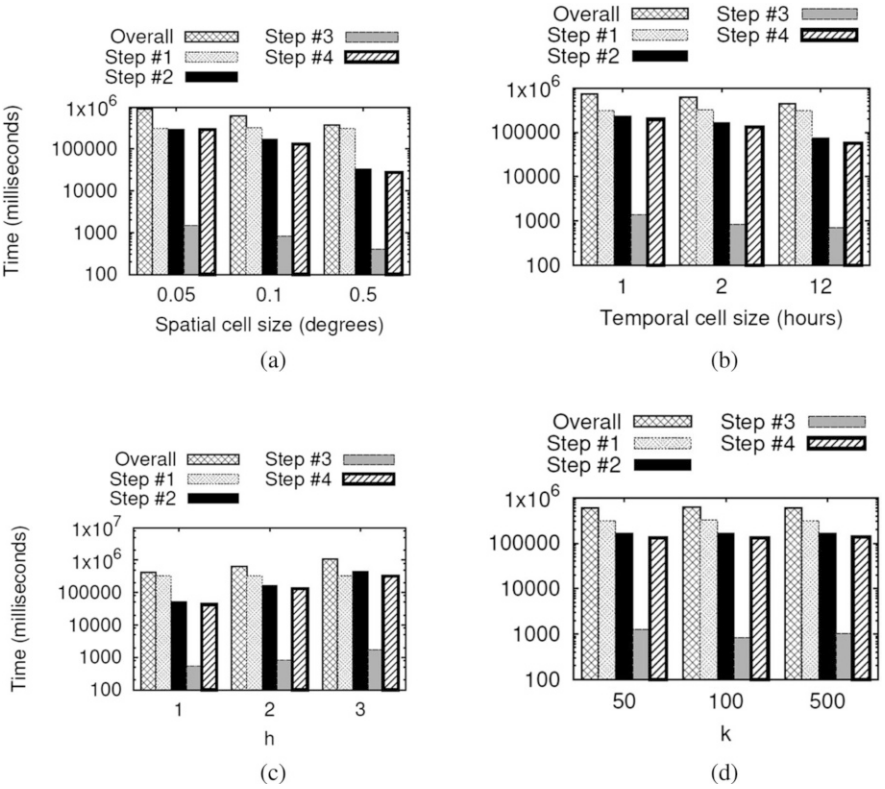


Fig. 10.9 Performance of our algorithm for various (**a**) spatial cell sizes of P, (**b**) temporal cell sizes of P, (**c**) values of h, and (**d**) values of k

10.4.4.1 Experimental Results

Varying the Spatial Cell Size In Fig. 10.9a, we demonstrate the results of our experiments by varying the spatial cell size of P. Higher sized spatial cells decrease the total number of cells n used in the grid partitioning of the 3D space. In turn, this is expected to lead to reduced execution time, since fewer cells need to be computed and lower communication is required by the algorithm. Indeed, the overall execution time is reduced when the grid is of coarser granularity.

Varying the Temporal Cell Size Figure 10.9b demonstrates the efficiency of our algorithm for various temporal cell sizes. The effect of larger cells in the temporal dimension to the overall execution time is similar to the previous experiment: Larger temporal cell size leads to fewer total cells in the grid, thus reducing the overall execution time. The experiment with the most coarse temporal partitioning (12 h) was measured to be twice more efficient than the experiment using the finest partitioning, in total execution time.

Varying h $aTHS$ can be parameterized with a user-defined variable h, which defines the set of neighboring cells contributing to the calculation of each cell's z-score. Figure 10.9c demonstrates the experimental results when using different values for variable h. The overall execution time is significantly reduced for lower values of h, since each cell broadcasts its attribute value to less neighboring cells, thus reducing the network overhead for exchanging such information between cells. By using a value of 3 for variable h, we measured three times higher overall execution time compared to the experiment having a value of 1. This significant reduction to the total execution time in $aTHS$, results in an approximate result $\hat{G}_i^*$. However, the deviation of $\hat{G}_i^*$ to G_i^* can be quantified as already explained.

Varying Top-k The value of top-k affects the size of the final result set. Figure 10.9d demonstrates the impact of this variable on dataset sizes throughout the execution of our algorithm and the individual steps' processing times. The overall execution time is not significantly affected by the value of this variable.

10.5 Distributed Data-Enriched Mobility Networks

Inference of the underlying network, given a large number of moving traces, both in aviation and maritime domain, is a challenging task that we try to address. The goal is to discover the directed graph of transitions, i.e., the set V of vertices and the set E of edges that form the routes network. Additionally, enriched information has to be taken into account in order to produce enriched graphs with contextual information. Domain experts may benefit a lot from such additional information. For example, one can then easily produce analytics of trajectories based on specific weather conditions and reveal how these conditions affecting or not the paths followed. Moreover, flight plans or predefined sea routes can be compared with real paths followed by ships or planes and the domain expert would be able to identify and explain the reason more easily.

10.5.1 Definitions

Definition 10.1 (Enriched Point) An enriched point r_i corresponds to a (raw point) p_i of a moving object and is defined as a tuple $< p_i, t_i, v_i >$, where v_i is a multi-dimensional vector consisting of categorical and/or numerical variables that annotate the raw point with associated context data.

Examples of v_i attribute values could be any user-defined tag or annotation valid regarding the specific domain application (e.g., consider annotations made by an event recognition module that detects the "top-of-climb" or "top-of-descent," "stop,"

"turn," etc.) or any numerical variable that can be attached to p_i, such as weather information (e.g., temperature, wind speed, humidity, etc.).

Definition 10.2 (Semantic Trajectory) An enriched trajectory R corresponds to a (raw) trajectory T of a moving object, which is defined as the sequence of the enriched points of T.

Definition 10.3 (Data-Enriched Mobility Network) A data-enriched mobility network N is a graph denoted by $N = (V, E)$, where V is a set of vertices and $E \subseteq V \times V$ is a set of edges.

The set V of vertices corresponds to the union of sets of enriched points where each set of enriched points is of the same enriched category, while the set E of edges corresponds to the union of paths in between vertices found. Given the above definitions the problem can now be formally expressed.

Problem 10.6 (Data-Enriched Network Inference) Given a database of enriched trajectories, infer the underlying data-enriched network.

The data-enriched network that is to be found should meet the following requirements, in order to provide added value to domain experts:

1. **Consistency**. Network vertices inferred ideally should belong to one connected component, or the network must be of a number of connected components.
2. **Node Validity**. Vertices of the network correspond to enriched categories and are valid if derived from an aggregation or an assemblage (depending on the corresponding method that is used) of a number of more than m enriched points, thus having real value for the domain experts. For example, if a network vertex is derived from less than m points, then this vertex might be excluded from the final vertices set.
3. **Edge Validity**. Edges must correspond to frequent paths followed by objects. The frequency of paths is determined as defined by an application-dependent threshold σ.

10.5.2 Discovering Data-Enriched Mobility Networks

We follow an approach where first the vertices are discovered and then edges connecting these vertices are inferred from the trajectories. The input to the process comprises of a set of enriched trajectories. An enriched trajectory is modeled as a sequence of timestamped enriched points. The output of the process is a semantic-aware mobility network modeled as a directed graph $G = (V, E)$, where the vertices V correspond to semantic nodes and the edges E correspond to the discovered paths between semantic nodes. The process comprises two main steps, described subsequently in detail and illustrated in Fig. 10.10a–d.

The network discovery method described above is applicable to both domains, maritime and aviation, although the set of enriched trajectories which is the input to

the process is formed differently for each domain. In the maritime domain the input is comprised of trajectories made by one vessel and covers a large temporal time frame. Thus, one maritime network is discovered for each vessel. Similarly, in the aviation domain the input is comprised of trajectories of aircraft that flew between two given airports. This way, one network is computed for every pair of airports. Note that the network is a directed graph, meaning that, e.g., Madrid–Barcelona is treated differently than the Barcelona–Madrid, implying that flights of only one direction at a time are taken for each pair of airports.

10.5.2.1 Step 1: Enriched Nodes Extraction

The input dataset of enriched trajectories is efficiently managed according to the enriched points and the enriched information each trajectory carries. By exploiting that each enriched point belongs to a semantic category (i.e., points carry an application depended tag, e.g., HOME, SPORT, etc., and these tags then make up the categories), we can split the input into enriched points of the same category. For every such input, nodes are formed based on a spatial-only clustering algorithm. Then, each cluster becomes a network node, carrying also semantic information from the corresponding category (Fig. 10.11). Moreover, the membership of a node in its corresponding cluster is above a given threshold m, thus having real value for the domain experts (i.e., suppress outliers).

Additionally, for each cluster/node several statistics are calculated and used later to refine and enhance the prediction accuracy. Statistics are modeled as Normal distributions (mean and standard deviation). Timing distributions describe when a ship or aircraft traverses the corresponding cluster/node within a day (24-h time frame). Elapsed time statistics describe the duration of staying in each cluster/node and speed statistics provide the mean speed passing through the corresponding cluster/node. Note that statistics may well be calculated after the discovery of the network as a post-processing step.

10.5.2.2 Step 2: Enriched Paths Discovery

Trajectories are processed separately of each other to identify enriched nodes and edges of the network, resulting in the discovery of semantic paths. More specifically, each cluster/node (i.e., corresponding enriched points of a moving object that clustered together) is processed separately and sequentially, ordered in the temporal dimension. If any two consecutive trajectory points correspond to two different network nodes/clusters, then a transition is recorded from one network node to the other. For each such transition, the beginning (from) and end (to) nodes of the network are identified and marked as additional information of the transition. These transitions form the set of candidate network edges.

Moreover, multiple transitions between pairs of network nodes are recorded by increasing (+1) the corresponding weight of the edge. In the end of the process,

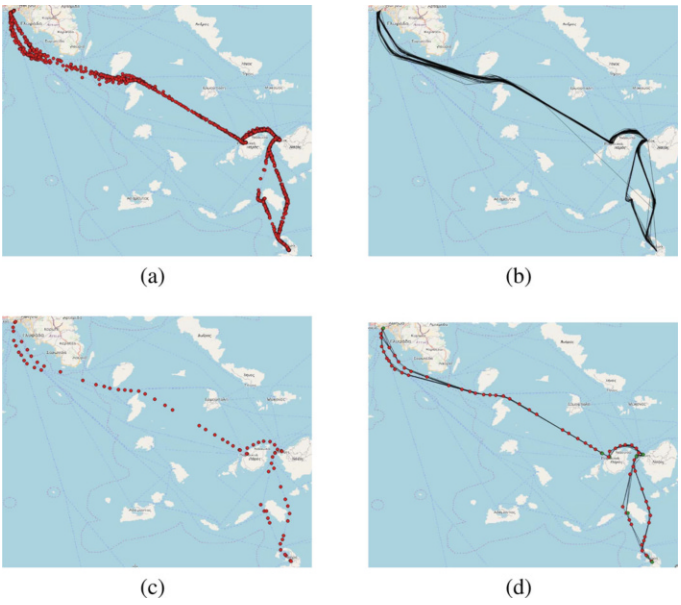


Fig. 10.10 Overview of semantic-aware network inference solution in maritime domain: (**a**) all enriched points from input, (**b**) enriched trajectories formed from enriched points, (**c**) enriched nodes extraction, (**d**) enriched path discovery

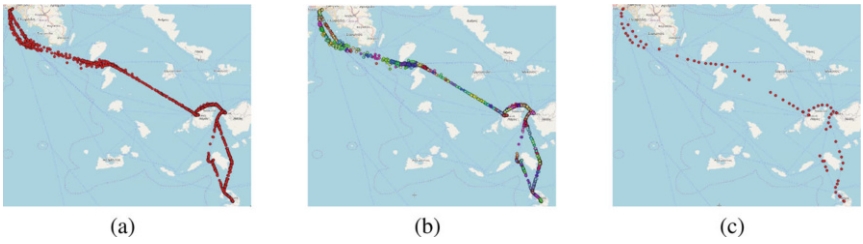
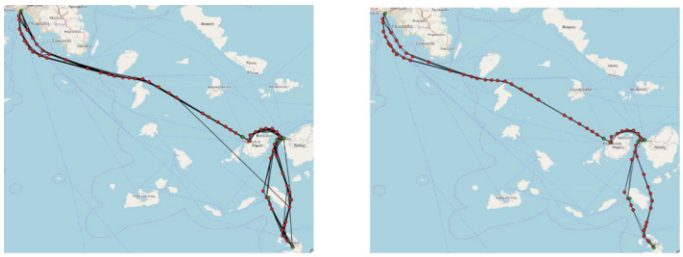


Fig. 10.11 Overview of network nodes extraction step in maritime domain: (**a**) all enriched points from input, (**b**) enriched points clustered spatially to candidate nodes, (**c**) semantic nodes extraction

all the edges/transitions are identified, along with their weights, which are simply the cardinality (absolute number) of all trajectories from the same edge. As an optional post-processing step, a threshold filtering can be applied by the domain expert if needed, in order to keep routes only above a specific weight or support. Finally, semantic nodes and edges are assembled to produce the complete map of the network. The weighted-edge network allows us to create a hierarchy of networks based on filtering edges with weight less than an application defined threshold σ. An example of paths discovery for maritime is shown in Fig. 10.12.



(a) (b)

Fig. 10.12 Overview of network paths discovery step in maritime domain: (**a**) all paths found, (**b**) only edges with more than σ weight are kept

Algorithm 10.4 SeaAirNet

Input: A semantic trajectory database STD, node membership m
Output: A semantic-aware spatial network N

1: **for** $e \in STD$ **do**
2: /* e is the category of points (CP) in STD */
3: $ClP \leftarrow SpatialClustering(CPe)$
4: /* ClP are tuples of the form $< clid, oid, p_i, t_i, e >$ */
5: **for** $ClP_i \in ClP$ **do**
6: **if** $ClP_i \neq Noise \wedge |ClP_i| > m$ **then**
7: $N.V \leftarrow N.V \cup createNode(ClP_i)$
8: $ClP_s \leftarrow ClP_s \cup ClP_i$
9: $N.E \leftarrow discoverEdges(ClPs, N.V)$
10: **return** N

10.5.3 The SeaAirNet Algorithm

We presented the steps of our methodology from an abstract point of view, accompanied with a running example and respective figures. Next, we present these steps in an algorithmic view. The main algorithm, named *SeaAirNet*, is the starting point.

Algorithm *SeaAirNet* takes as input a semantic trajectory database (STD) and a node membership limit m and outputs a semantic-aware mobility network. In line 1 all enriched points of the STD are partitioned based on the category each one belongs (semantic category) and then each partition is processed separately. In line 3 a spatial clustering algorithm is utilized to cluster enriched points of the specific category. For each cluster found (line 5), if the cluster is not considered as "noise" and its membership is over the application-dependent limit m (line 6), then it is considered valid and a network node is created (line 7) along with its statistics. It must be noted that the clustering algorithm might identify some points as noise: These do not belong in any cluster, or they do belong to a non-valid cluster depending on the clustering algorithm. In line 5 the newly created node is added to the set of network nodes. In line 8 each valid cluster is added to the set of valid

Algorithm 10.5 discoverEdges

Input: Clusters of enriched points ClP_s, network nodes V
Output: Network edges E

1: $nodesAndPoints \leftarrow CIP_s$.join V on $(clid, e)$
2: $P \leftarrow partition(nodesAndPoints$ on $oid).orderBy(t)$
3: **for** each $P_i \in P$ **do**
4: $E_i \leftarrow edge(getNode, getNextNode, 1)$
5: /* each E_i is of form $< fromNode,\ toNode,\ weight >$ */
6: **if** $E_i.node <> E_i.nextNode \wedge E_i.nextNode$ isValid **then**
7: $E \leftarrow E \cup E_i$
8: $E \leftarrow aggregate(E$ on $fromNode,\ toNode).sum(weight)$
9: **return** E

clusters that will later be used for the discovery of network edges. After extracting the semantic nodes (step 1), the paths discovery (step 2) task follows, where (in line 9) the *discoverEdges* algorithm is called by passing the set of valid clusters found and the set of network nodes extracted. Lastly, the returned nodes and edges (line 10) form the desired mobility network.

The algorithm *discoverEdges* discovers network-weighted edges from the clusters and network nodes found in algorithm *SeaAirNet*. Initially (line 1), variable *nodesAndPoints* holds a join of network nodes found (V) and corresponding enriched points that formed the node. In line 2, a partitioning technique is applied. In detail, the join set is partitioned by the object identifier (i.e., the id of vessel or aircraft), while the data of each partition is ordered based on time (t). Now, every partition holds one semantic trajectory with its enriched points ordered in time and also, each enriched point holds the cluster (network) node it belongs. For each such partition (line 3), enriched points are scanned sequentially in line 4 and every two consecutive points form an edge with weight of 1. Validity of the edge is checked in line 6: To be valid, the two network nodes it connects must be different and also the last node must be valid too, due to the sequential scanning of enriched points. If the edge is a valid network edge, then it is considered a candidate edge (line 7) and is added to the set of edges. When all partitions are processed, then results are aggregated on weight w.r.t. the beginning and ending nodes of each edge.

10.5.4 Experimental Results

In this section, we evaluate our approach of computing semantic-aware mobility networks both on aviation and on maritime domains. Our approach in this study is qualitative, meaning that we evaluate the produced networks by visually inspecting whether the network provides an accurate representation of the data used to extract it. We implemented all algorithms in Scala programming language, using Apache Spark 2.2.0 API. We implement a grid-density-based clustering algorithm in Spark to avoid the need of predefining the number of clusters in the beginning.

All algorithms are packed in one Scala project and when necessary, intermediate results are stored and retrieved using a database to avoid long lineages. Parameters of the algorithms are:

- *epsilon*: the radius of the disk that defines the neighborhood of a point;
- *minPts*: minimum number of members to consider a neighborhood as a valid cluster;
- m: node membership to consider a cluster as a valid network node;
- σ: threshold to keep edges of a certain weight and above in post-processing.

10.5.4.1 Datasets

In the maritime domain we selected to explore the movement of the DELOS vessel. DELOS has an mmsi of 241087000 and the spatiotemporal extend of its data is set to: $extend[x, y, t] = [23.61 \rightarrow 25.43, 36.39 \rightarrow 37.95, 2016-01-0100{:}11{:}34 \rightarrow 2016-01-3121{:}46{:}28]$. The above extent consists of 5761 raw points from IMIS Global AIS messages concerning specific vessel. These raw points are passed from synopses generator (SG) (cf. Chap. 3 of this book), which produced 2195 synopses whose enriched points are used for discovering the network.

In the aviation domain the MADRID (MAD)–BARCELONA (BCN) airports pair is selected using IFS radar surveillance data provided by CRIDA. The temporal extent is set to 2016-04-01 05:16:54 until 2016-04-30 20:06:08. The dataset consists of 997,450 raw points (both directions) which are derived from 1396 flights. The number of corresponding synopses produced by the SG are 254,330.

The methodology of discovering semantic-aware networks can be applied to raw trajectories as well as to semantic trajectories. Due to space limitations in the following we show an application of the methodology to raw trajectories in the aviation domain and an application to enriched trajectories in the maritime domain.

10.5.4.2 Qualitative Results in Aviation

We apply algorithm to raw 447,234 points of MAD to BCN airport pair (one direction), with parameters: $epsilon = [10000, 10000, 1000]$, $minPts = 100$, $m = 200$. We get 54 clusters (Fig. 10.13).

(a) (b)

Fig. 10.13 All points colored by cluster; (**a**) with noise (19,882) and (**b**) without noise

These clusters are transformed to nodes by keeping only the medoid of each cluster. We consider only clusters with more than 200 members; thus we get 39 nodes (Fig. 10.14).

Then edges are discovered between above nodes. In total we found 182 edges with weights from 1 to 1274 (average 89.51). Based on domain expert, edges can be filtered out using their weight (Fig. 10.15).

Fig. 10.14 Medoids of the 39 clusters found (**a**) and how medoids cover the whole dataset (**b**)

Fig. 10.15 Discovering paths (edges) of the network, (**a**) all edges found with weight over 0 (182), (**b**) keep only edges with weight over 2 (130)

10.5.4.3 Qualitative Results in Maritime

We applied our algorithm to the 2195 points of trajectories' synopses of DELOS in the Aegean Sea, with parameters $epsilon = [1000, 1000, 100]$, $minPts = 1$, $m = 5$. Synopses are grouped in two groups: either as stops and/or as changes (Fig. 10.16).

These clusters are transformed to nodes, where each node represents one cluster. In total we get 128 nodes with membership from 1 to 224 (average 19.84). We consider only clusters with more than 5 members and thus we get 73 nodes (Fig. 10.17).

Then edges are discovered between these nodes. The algorithm finds 359 edges with weights from 1 to 91 (average 4.98). Figure 10.18 shows network edges for various values of σ.

From the above qualitative evaluation, it can be concluded that the higher the weight of the edges, the higher the compactness in the representation of the

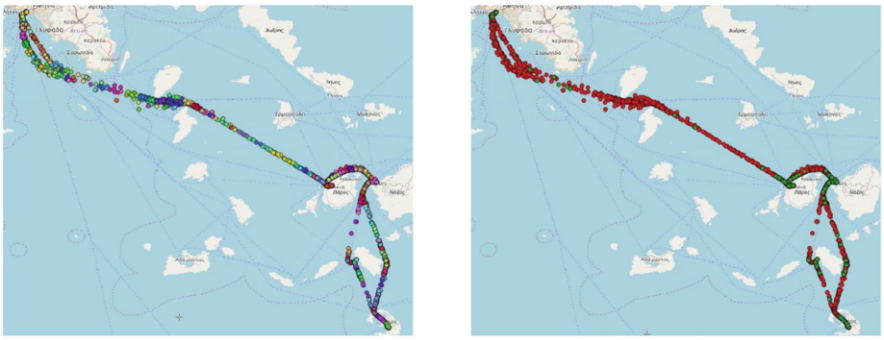


Fig. 10.16 All points colored by cluster above (**a**) and grouped based on their semantics in (**b**)

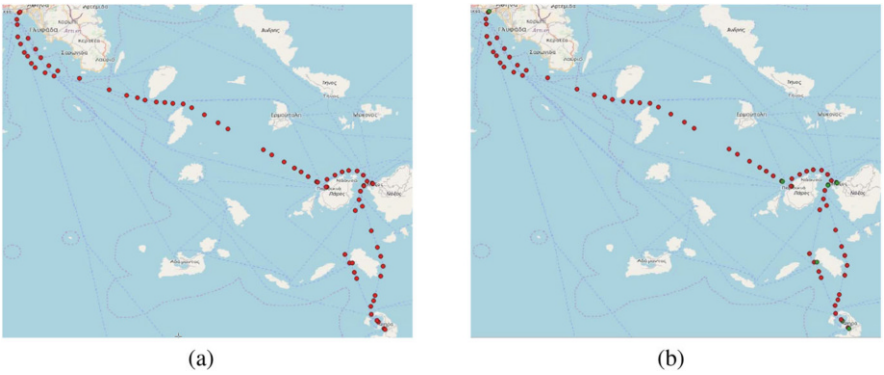


Fig. 10.17 Medoids of the 73 clusters found (**a**) and colored by type in (**b**)

dataset with this data-enriched network structure. Also, the network extracted from synopses is more or less the same as the one produced by the raw data. The advantage of this is that not only we may extract the network by processing much less data, but more importantly, we exploit the synopses to attach semantics to the vertices of the network.

10.6 Related Work

In recent years, an increased research interest has been observed in knowledge discovery out of mobility data. Towards this direction, several methods, which are directly related to our work, have been proposed.

Co-movement Patterns One of the first approaches for identifying such collective mobility behavior is the so-called flock pattern [14]. Inspired by this, a less "strict"

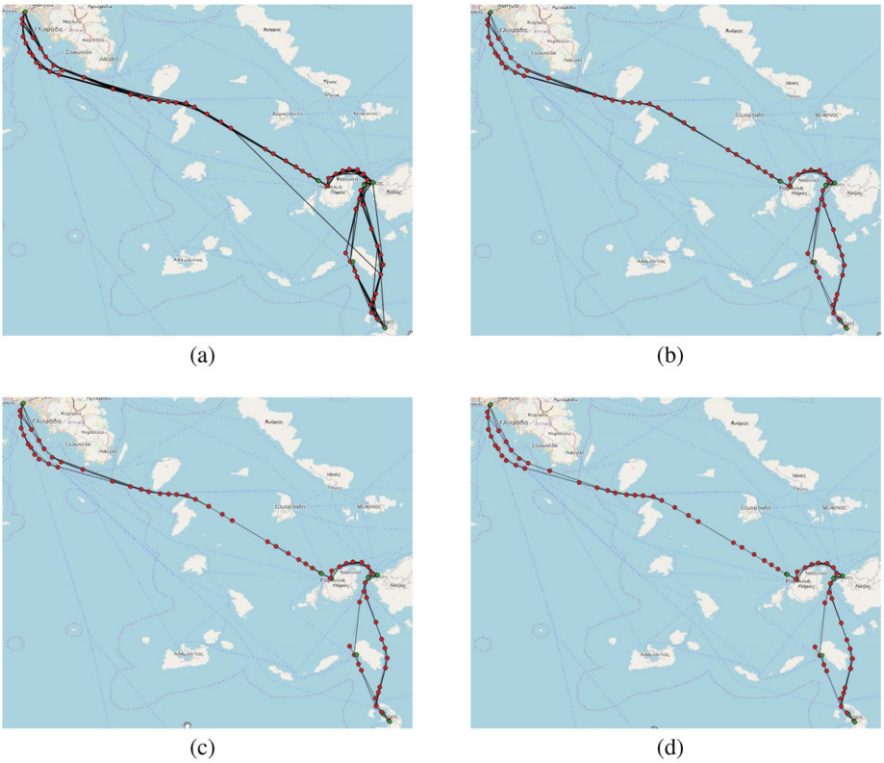


Fig. 10.18 Discovering paths (edges) of the network, (**a**) all edges found with weight over 0 (359), (**b**) edges with weight over 1 (226), (**c**) edges with weight over 2 (176), (**d**) edges with weight over 3 (136)

definition of flocks was proposed in [12] where the notion of a moving cluster was introduced. There are several related works that emerged from the above ideas, like the approaches of convoys, swarms, platoons, traveling companion, and gathering pattern [39]. However, all of the aforementioned approaches are centralized and cannot scale to massive datasets. In this direction, the problem of efficient convoy discovery was studied both in centralized [22] and distributed environment by employing the MapReduce programming model [21]. An approach that defines a new generalized mobility pattern is presented in [9]. In more detail, the general co-movement pattern (GCMP) is proposed, which models various co-movement patterns in a unified way and is deployed on a modern distributed platform (i.e., Apache Spark) to tackle the scalability issue.

Trajectory Clustering Most of the aforementioned approaches operate at specific predefined temporal "snapshots" of the dataset, thus ignoring the route of each moving object between these "snapshots." Another line of research tries to discover groups of either entire or portions of trajectories considering their routes. A

typical strategy in dealing with trajectory clustering is to transform trajectories to a multi-dimensional space and then apply well-known clustering algorithms such as OPTICS [2] and DBSCAN [8]. Alternatively, another approach is to define an appropriate similarity function and embed it to an extensible clustering algorithm. In this direction, there are several approaches whose goal is to group whole trajectories, including T-OPTICS [19], that incorporates a trajectory similarity function into the OPTICS [2] algorithm. CenTR-I-FCM [26], a variant of Fuzzy C-means, proposes a specialized similarity function that aims to tackle the inherent uncertainty of trajectory data. Nevertheless, trajectory clustering is a computationally intensive operation and centralized solutions cannot scale to massive datasets. In this context, [6] introduces a scalable GPU-based trajectory clustering approach which is based on OPTICS [2].

Subtrajectory Clustering Nonetheless, discovering clusters of complete trajectories can overlook significant patterns that might exist only for portions of their lifespan. To deal with this, another line of research has emerged, that of *Subtrajectory Clustering*. The predominant approach here is TraClus [15], a partition-and-group framework for clustering 2D moving objects (i.e., the time dimension is ignored) that enables the discovery of common subtrajectories. A more recent approach to the problem of subtrajectory clustering is S^2T-Clustering [28], where the goal is to partition trajectories into subtrajectories and then form groups of similar ones, while, at the same time, separate the ones that fit into no group, called outliers. It consists of two phases: a neighborhood-aware trajectory segmentation (*NaTS*) phase and a sampling, clustering, and outlier (*SaCO*) detection phase. A slightly different approach is presented in QuT-Clustering [27] and [33], where the goal is, given a temporal period of interest W, to efficiently retrieve the clusters and outliers at subtrajectory level that temporally intersect W. In order to achieve this, a hierarchical structure, called ReTraTree (for Representative Trajectory Tree) that effectively indexes a dataset for subtrajectory clustering purposes, is built and utilized. An alternative viewpoint to the problem of subtrajectory clustering is presented in [1], where the goal is to identify "common" portions between trajectories, w.r.t. some constraints and/or objectives, cluster these "common" subtrajectories, and represent each cluster as a pathlet, which is a point sequence that is not necessarily a subsequence of an actual trajectory. A pathlet can be viewed as a portion of a path that is traversed by many trajectories. Similarly, in [40] the goal is to identify corridors, which are frequent routes traversed by a significant number of moving objects. As already mentioned, all of the above subtrajectory clustering approaches are centralized and cannot scale to the size of today's trajectory data.

Hotspot Analysis The problem of trajectory hotspot analysis is related to the spatial and spatiotemporal hotspot analysis. Several studies exist for conducting hotspot spatiotemporal analysis, such as [11, 17]. Spatiotemporal event data are analyzed and visualized in [17]. It consists of two steps: first, it uses multivariate kernel density estimation in space and time to estimate the density of the input data. Interestingly, different kernels in spatial and temporal domains can be used. In the second step it identifies hotspots from the most dense kernels and proposes

a new visualization technique, based on Reeb graphs to illustrate the identified spatiotemporal hotspots. Hong et al. [11] studied the case of human mobility data such as taxi trips, bike rides, and subway trips. This data can be modeled by using spatiotemporal directed graphs (STG). The goal is to find subgraphs of the STG that have interesting flows (e.g., a black hole has the overall inflow greater than the overall outflaw). The user needs to input a threshold in order for the algorithm to successfully identify the interesting flows. A similar study is presented in [13] where human mobile traffic data are analyzed to discover hotspots on graphs. This study identifies spatial locations where the data volumes from wireless network transmissions are unusually high, based on user-defined thresholds. After identifying these locations, the algorithm detects the distribution of mobile data traffic hotspots to propose an efficient cell deployment strategy.

The trajectory hotspot analysis problem is also related to the trajectory mining domain [39]. Such trajectory and subtrajectory clustering techniques have been presented previously, in this section.

Data-Enriched Network Discovery Several methods rely on k-means clustering of raw tracking data using distance and direction as criteria to introduce cluster seeds at fixed locations along a trajectory. Edelkamp and Schroedl [7] use various heuristics for segmentation, map matching, and lane clustering from GPS traces. Schroedl et al. [30] use k-means clustering to refine an existing network map rather than construct the entire network starting from a blank terrain. Other methods transform GPS traces to density-based discretized images and are based on kernel density estimation (KDE). Most of these algorithms function well either when the data are frequently sampled (e.g., once per second) or when there is a lot of data redundancy. Biagioni and Eriksson [3] use a dataset which is being sampled very frequently (from 2 to 6 s). Steiner and Leonhardt [32] present an approach which uses tracking data of lower frequency, but still with intervals not exceeding 15 s. The limitation of KDE-based algorithms is that they are quite sensitive with respect to noisy data and outliers.

Another category, to which the present work relates, involves trace clustering approaches. These methods either adopt map matching or heuristics by aggregating GPS traces into an incrementally built transportation network. Moving object's heading and distance measures are also used to perform additions and deletions onto the incremental construction of the map. Rogers et al. [29] use trace clustering to refine an existing network rather than extracting it from scratch. Cao and Krumm [4] eliminate noise in GPS traces, while Fathi and Krumm [10] provide an approach that discovers intersections by using a prototypical detector trained on ground truth data from an existing map. This approach works best for well-aligned transportation networks (e.g., vertically aligned road networks) and with frequently sampled data of up to 5 s. Liu et al. [16] efficiently build a map but require accurate data and high sampling rates (i.e., 1 s). Zhang et al. [38] use a method similar to GPS trace clustering to continuously refine existing maps.

In general, although the problems of map construction, update, and enhancement are complementary, typically each individual work focuses on a single one of them. For example, a recent work by Wang et al. [37] applies trace clustering techniques to introduce a new KDE-based road fitting algorithm. The authors achieve an important contribution in terms of map entries in terms of data records on the OpenStreetMap collection, but their application mainly focuses on updating a map rather than constructing it. Similarly, Shan et al. [31] extend by proposing an automatic map update system which focuses on the identification of missing segments and is robust w.r.t. low sampling rates (on average of 120 s). Wang et al. [37] efficiently tackle the hard time performance of current approaches, deal with tracking data of low sampling rate but they mainly focus on inferring a map attributed with topological characteristics.

10.7 Discussion: Lessons Learnt

In this chapter, we reported on offline data analytic methods over moving object trajectories. The overall objective was to develop advanced, beyond current state-of-the-art data analytics methods and tools over a repository of trajectories of moving objects. In detail, we initially studied the problem of trajectory clustering by utilizing a methodology, which transforms trajectories to vectors, so as existing big-data-ready, point-based clustering algorithms (such as those provided by the Spark MLlib machine learning library) can be used. The goal of this approach is on the one hand to first provide solutions for the whole-trajectory clustering problem and to study the limitations of using off-the-shelf clustering algorithms for big trajectory data. Subsequently, the problem of distributed (sub)trajectory clustering over massive mobility data [35] has been addressed. In order to provide a solution to this we build upon a scalable distributed trajectory join method [34], which utilizes the popular MapReduce distributed programming model. Successively, we presented a scalable distributed trajectory-based hotspot analysis [20]. In this line of research, we followed a different clustering approach that provides statistical guarantees for the identified clusters. Interestingly, as a proof of concept, with this approach, we are able to discover hotspots that in the maritime domain can be used to measure the fishing pressure at sea, while in the aviation domain it identifies air-blocks that present demand-capacity problems. Finally, we studied the problem of distributed data-enriched mobility network discovery. The algorithms proposed provide contextually enhanced spatial graphs, which can successfully be utilized to support online location and trajectory prediction/forecasting scenarios (cf. Chap. 8).

References

1. Agarwal, P.K., Fox, K., Munagala, K., Nath, A., Pan, J., Taylor, E.: Subtrajectory clustering: models and algorithms. In: PODS, pp. 75–87 (2018)
2. Ankerst, M., Breunig, M.M., Kriegel, H., Sander, J.: OPTICS: ordering points to identify the clustering structure. In: SIGMOD, pp. 49–60 (1999)
3. Biagioni, J., Eriksson, J.: Map inference in the face of noise and disparity. In: Proceedings of the 20th International Conference on Advances in Geographic Information Systems, pp. 79–88 (2012)
4. Cao, L., Krumm, J.: From GPS traces to a routable road map. In: Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, pp. 3–12 (2009)
5. Claramunt, C., Ray, C., Camossi, E., Jousselme, A., Hadzagic, M., Andrienko, G.L., Andrienko, N.V., Theodoridis, Y., Vouros, G.A., Salmon, L.: Maritime data integration and analysis: recent progress and research challenges. In: Proceedings of the 20th International Conference on Extending Database Technology, EDBT, pp. 192–197 (2017)
6. Deng, Z., Hu, Y., Zhu, M., Huang, X., Du, B.: A scalable and fast OPTICS for clustering trajectory big data. Clust. Comput. **18**(2), 549–562 (2015)
7. Edelkamp, S., Schrödl, S.: Route Planning and Map Inference with Global Positioning Traces, pp. 128–151. Springer, Berlin (2003)
8. Ester, M., Kriegel, H., Sander, J., Xu, X.: A density-based algorithm for discovering clusters in large spatial databases with noise. In: KDD, pp. 226–231 (1996)
9. Fan, Q., Zhang, D., Wu, H., Tan, K.: A general and parallel platform for mining co-movement patterns over large-scale trajectories. Proc. VLDB Endowment **10**(4), 313–324 (2016)
10. Fathi, A., Krumm, J.: Detecting road intersections from GPS traces. In: Geographic Information Science, pp. 56–69 (2010)
11. Hong, L., Zheng, Y., Yung, D., Shang, J., Zou, L.: Detecting urban black holes based on human mobility data. In: Proceedings of the 23rd International Conference on Advances in Geographic Information Systems SIGSPATIAL, pp. 35:1–35:10 (2015)
12. Kalnis, P., Mamoulis, N., Bakiras, S.: On discovering moving clusters in spatio-temporal data. In: SSTD, pp. 364–381 (2005)
13. Klessig, H., Suryaprakash, V., Blume, O., Fehske, A.J., Fettweis, G.: A framework enabling spatial analysis of mobile traffic hot spots. IEEE Wirel. Commun. Lett. **3**(5), 537–540 (2014). https://doi.org/10.1109/LWC.2014.2349520
14. Laube, P., Imfeld, S., Weibel, R.: Discovering relative motion patterns in groups of moving point objects. Int. J. Geogr. Inf. Sci. **19**(6), 639–668 (2005)
15. Lee, J., Han, J., Whang, K.: Trajectory clustering: a partition-and-group framework. In: SIGMOD, pp. 593–604 (2007)
16. Liu, X., Biagioni, J., Eriksson, J., Wang, Y., Forman, G., Zhu, Y.: Mining large-scale, sparse GPS traces for map inference: comparison of approaches. In: Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 669–677 (2012)
17. Lukasczyk, J., Maciejewski, R., Garth, C., Hagen, H.: Understanding hotspots: a topological visual analytics approach. In: Proceedings of the 23rd International Conference on Advances in Geographic Information Systems SIGSPATIAL, pp. 36:1–36:10 (2015)
18. Moran, P.: Notes on continuous stochastic phenomena. Biometrika **37**(1), 17–23 (1950)
19. Nanni, M., Pedreschi, D.: Time-focused clustering of trajectories of moving objects. J. Intell. Inf. Syst. **27**(3), 267–289 (2006)
20. Nikitopoulos, P., Paraskevopoulos, A., Doulkeridis, C., Pelekis, N., Theodoridis, Y.: Hot spot analysis over big trajectory data. In: IEEE International Conference on Big Data, Big Data 2018, Seattle, WA, 10–13 December 2018, pp. 761–770 (2018). https://doi.org/10.1109/BigData.2018.8622376
21. Orakzai, F., Calders, T., Pedersen, T.B.: Distributed convoy pattern mining. In: IEEE MDM, pp. 122–131 (2016)

22. Orakzai, F., Calders, T., Pedersen, T.B.: k/2-hop: fast mining of convoy patterns with effective pruning. Proc. VLDB Endowment **12**(9), 948–960 (2019)
23. Ord, J.K., Getis, A.: Local spatial autocorrelation statistics: distributional issues and an application. Geogr. Anal. **27**(4), 286–306 (1995)
24. Panagiotakis, C., Tziritas, G.: A speech/music discriminator based on RMS and zero-crossings. IEEE Trans. Multimedia **7**(1), 155–166 (2005)
25. Panagiotakis, C., Kokinou, E., Vallianatos, F.: Automatic p-phase picking based on local-maxima distribution. IEEE Trans. Geosci. Remote Sens. **46**(8), 2280–2287 (2008)
26. Pelekis, N., Kopanakis, I., Kotsifakos, E.E., Frentzos, E., Theodoridis, Y.: Clustering uncertain trajectories. Knowl. Inf. Syst. **28**(1), 117–147 (2011)
27. Pelekis, N., Tampakis, P., Vodas, M., Doulkeridis, C., Theodoridis, Y.: On temporal-constrained sub-trajectory cluster analysis. Data Min. Knowl. Discov. **31**(5), 1294–1330 (2017)
28. Pelekis, N., Tampakis, P., Vodas, M., Panagiotakis, C., Theodoridis, Y.: In-DBMS sampling-based sub-trajectory clustering. In: EDBT, pp. 632–643 (2017)
29. Rogers, S., Langley, P., Wilson, C.: Mining GPS data to augment road models. In: Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 104–113 (1999)
30. Schroedl, S., Wagstaff, K., Rogers, S., Langley, P., Wilson, C.: Mining GPS traces for map refinement. Data Min. Knowl. Discov. **9**, 59–87 (2004)
31. Shan, Z., Wu, H., Sun, W., Zheng, B.: Cobweb: a robust map update system using GPS trajectories. In: Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing, pp. 927–937 (2015)
32. Steiner, A., Leonhardt, A.: A map generation algorithm using low frequency vehicle position data contents. In: 90th Annual Meeting of the Transportation Research Board (2011)
33. Tampakis, P., Pelekis, N., Andrienko, N.V., Andrienko, G.L., Fuchs, G., Theodoridis, Y.: Time-aware sub-trajectory clustering in hermes@postgresql. In: ICDE, pp. 1581–1584 (2018)
34. Tampakis, P., Doulkeridis, C., Pelekis, N., Theodoridis, Y.: Distributed subtrajectory join on massive datasets. ACM Trans. Spatial Algorithms Syst. **6**(2) (2019). https://doi.org/10.1145/3373642
35. Tampakis, P., Pelekis, N., Doulkeridis, C., Theodoridis, Y.: Scalable distributed subtrajectory clustering. In: IEEE BigData 2019, pp. 950–959 (2019)
36. Vlachos, M., Gunopulos, D., Kollios, G.: Discovering similar multidimensional trajectories. In: ICDE, pp. 673–684 (2002)
37. Wang, S., Wang, Y., Li, Y.: Efficient map reconstruction and augmentation via topological methods. In: Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems, pp. 25:1–25:10 (2015)
38. Zhang, L., Thiemann, F., Sester, M.: Integration of GPS traces with road map. In: Proceedings of the Third International Workshop on Computational Transportation Science, pp. 17–22 (2010)
39. Zheng, Y.: Trajectory data mining: an overview. ACM Trans. Intell. Syst. Technol. **6**(3), 29:1–29:41 (2015)
40. Zygouras, N., Gunopulos, D.: Corridor learning using individual trajectories. In: IEEE MDM, pp. 155–160 (2018)