Hot Spot Analysis over Big Trajectory Data

Panagiotis Nikitopoulos[1], Aris-Iakovos Paraskevopoulos[2], Christos Doulkeridis[3], Nikos Pelekis[4] and Yannis Theodoridis[5]

School of Information and Communication Technologies
University of Piraeus, 18534 Piraeus, Greece
{[1]*nikp,*[3]*cdoulk,*[4]*npelekis,*[5]*ytheod*}*@unipi.gr,* [2]*arisparaskevop@gmail.com*

***Abstract*—Hot spot analysis is the problem of identifying statistically significant spatial clusters from an underlying data set. In this paper, we study the problem of hot spot analysis for massive trajectory data of moving objects, which has many real-life applications in different domains, especially in the analysis of vast repositories of historical traces of spatio-temporal data (cars, vessels, aircrafts). In order to identify hot spots, we propose an approach that relies on the Getis-Ord statistic, which has been used successfully in the past for point data. Since trajectory data is more than just a collection of individual points, we formulate the problem of *trajectory hot spot analysis*, using the Getis-Ord statistic. We propose a parallel and scalable algorithm for this problem, called THS, which provides an exact solution and can operate on vast-sized data sets. Moreover, we introduce an approximate algorithm (aTHS) that avoids exhaustive computation and trades-off accuracy for efficiency in a controlled manner. In essence, we provide a method that quantifies the maximum induced error in the approximation, in relation with the achieved computational savings. We develop our algorithms in Apache Spark and demonstrate the scalability and efficiency of our approach using a large, historical, real-life trajectory data set of vessels sailing in the Eastern Mediterranean for a period of three years.**

***Index Terms*—Hot spot analysis, trajectory data, parallel processing, MapReduce, Apache Spark**

I. Introduction

Huge amounts of mobility data is being produced at unprecedented rates everyday, due to the proliferation of GPS technology, the widespread adoption of smartphones, social networking, as well as the ubiquitous nature of monitoring systems. This data wealth contributes to the ever-increasing size of what is recently known as big spatial (or spatio-temporal) data [1], a specialized category of big data focusing on mobile objects, where the spatial and temporal dimensions have elevated importance. Such data include mobile objects' trajectories, geotagged tweets by mobile Twitter users, check-ins in Foursquare, etc. Analyzing spatio-temporal data has the potential to discover hidden patterns or result in non-trivial insights, especially when its immense volume is considered. To this end, specialized parallel data processing frameworks [2]–[5] and algorithms [6]–[9] have been recently developed aiming at spatial and spatio-temporal data management at scale.

In this context, a useful data analysis task is *hot spot analysis*, which is the process of identifying *statistically significant* clusters (i.e., clusters which have low probability values, based on a specific trajectory attribute). Motivated by the need for big data analytics over trajectories of vessels, we focus on discovering hot spots in the maritime domain, as this relates to various challenging use-case scenarios [10]. More specifically, having a predefined tessellation of a region into areas of interest for which there is a priori knowledge about occurring activities in them, it is very useful to be able to analyze – for instance – the intensity of the fishing activity (i.e., fishing pressure) of the areas, or to quantify the environmental fingerprint by the passage of a particular type of vessels from the areas. Similar cases exist in all mobility domains. In the aviation domain, the predicted presence of a number of aircrafts above a certain threshold results in regulations in air traffic, while in the urban domain such a presence accompanied with low speed patterns implies traffic congestions. Thus, the effective discovery of such diverse types of hot spots is of critical importance for our ability to comprehend the various domains of mobility.

Our approach for hot spot discovery and analysis is based on spatio-temporal partitioning of the 3D data space in cells. Accordingly, we try to identify cells that constitute hot spots, i.e., not only do they have high density, but also that the density values are statistically significant. We employ the Getis-Ord statistic [11], a popular metric for hot spot analysis, which produces z-scores and p-values by aggregating the density values of neighboring cells. A cell is considered as a hot spot, 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 spatio-temporal positions, and the applicability of hot spot 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 hot spot 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 hot spots in terms of spatio-temporal 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.

In summary, our work makes the following contributions:

- We formulate the problem of *trajectory hot spot analysis*, by means of the popular Getis-Ord statistic, appropriately tailored to become meaningful for sequences of spatio-temporal positions, rather than plain points.
- We present a parallel algorithm that provides an exact solution to the problem, and returns spatio-temporal cells with high scores that essentially constitute hot spots.
- To improve the efficiency, we also propose an approximate parallel algorithm and a method that is able to trade-off result accuracy for computational cost, that bounds the error of the approximation in a controlled way.
- We developed our algorithms in Apache Spark and we demonstrate their efficiency and scalability by experimental evaluation on a large data set of vessel trajectories in the Eastern Mediterranean Sea that span three years in total.

The remainder of this paper is structured as follows: Section II formulates the problem under study. Section III presents the proposed algorithm for hot spot analysis for big trajectory data along with its implementation details. In Section IV, we present the approximate algorithm that solves the problem with much lower processing cost, and with controlled error in the accuracy. Then, in Section V, we present the experimental results using real-life data set, and in Section VI we provide an overview of research related to hot spot analysis and trajectory data management. Finally, Section VII concludes the paper.

II. Problem Formulation

Consider a moving object database $\mathcal{D}$ that consists of trajectories $\tau \in \mathcal{D}$ of moving objects. A trajectory, denoted by an identifier τ, is a sequence of data points p described by 2D spatial coordinates ($p.x$ and $p.y$), a timestamp ($p.t$), as well as any other information related to the spatio-temporal position of p. For example, attributes referring to weather information. We also use $p.\tau$ to refer to the trajectory that p belongs to. Furthermore, consider a spatio-temporal partitioning $\mathcal{P}$ which partitions the 3D spatio-temporal domain into n 3D cells $\{c_1, \ldots, c_n\} \in \mathcal{P}$. Each data point p is mapped to one cell c_i, which is determined based on p being enclosed in c_i. 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: $x_i = \sum_{\tau \in c_i} \frac{t_{end} - t_{start}}{c_{i_{end}} - c_{i_{start}}}$, thus each trajectory τ that exists in a spatio-temporal cell c_i contributes to the cell's attribute value by its temporal duration $t_{end} - t_{start}$ in c_i 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 trajectory stays in a spatio-temporal cell, the higher its contribution to the cell's hot spot value.

Symbol	Description
$\mathcal{D}$	Spatio-temporal data set
$p \in \mathcal{D}$	Spatio-temporal data point ($p.x, p.y, p.t$)
τ	A trajectory (as a sequence) of points $p_1, p_2, \ldots$
$\mathcal{P}$	3D space partitioning $\mathcal{P} = \{c_1, \ldots, c_n\}$
c_i	The i-th cell of partitioning $\mathcal{P}$, ($1 \leq i \leq n$)
x_i	Attribute value of cell $c_i \in \mathcal{P}$
$w_{i,j}$	A weight indicating the influence of cell c_j to c_i
α	Parameter used to define the weights $w_{i,j}$
n	The number of cells in $\mathcal{P}$
G_i^*	The Getis-Ord statistic for cell c_i
$\tilde{G}_i^*$	An approximate value of G_i^*
$\mathcal{H}_{ij}$	Distance between cells c_i and c_j (in number of cells)
top-k	Requested number of most significant cells

TABLE I
Overview of symbols.

The problem of *hot spot analysis* addressed in this work is to identify statistically significant spatio-temporal areas (i.e., cells of $\mathcal{P}$), where the significance of a cell c_i is a function of the cell's attribute value x_i, but also of other neighboring cells' attribute values. A commonly used function (statistic) is the Getis-Ord statistic G_i^*, defined as [11]:

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

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

$$\overline{X} = \frac{\sum_{j=1}^{n} x_j}{n} \quad (2)$$

$$S = \sqrt{\frac{\sum_{j=1}^{n} x_j^2}{n} - (\overline{X})^2} \quad (3)$$

To model the intuition that the influence of a neighboring cell c_i to a given cell c_j should be decreasing with increased distance, we employ a weight function that decreases with increasing distance. Namely, we define: $w_{ij} = a^{1-\rho}$, where $a > 1$ is an application-dependent parameter, and ρ represents the distance between cell i and cell j measured in number of cells[1]. For immediate neighboring cells, where $\rho = 1$, we have $w_{ij} = 1$, while for the next neighbors we have respectively: $1/a$, $1/a^2$, ...,. This definition captures an "exponential decay" with distance in the contribution of neighboring cells to a given cell.

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

Problem 1: (*Trajectory hot spot analysis*) Given a trajectory data set $\mathcal{D}$ and a space partitioning $\mathcal{P}$, find the top-k cells $TOPK = \{c_1, \ldots, c_k\} \in \mathcal{P}$ based on the Getis-Ord statistic G_i^*, such that: $G_i^* \geq G_j^*$, $\forall c_i \in TOPK$, $c_j \in \mathcal{P} - TOPK$.

[1]Notice that this distance applies to the 3D grid, i.e., cells with the same spatial extent that belong to different temporal intervals have non-zero distance.

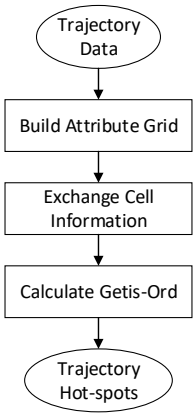


Fig. 1. Overview of THS algorithm.

In this paper, we study an instance of Problem 1, where the aim is to perform hot spot analysis for trajectories over massive spatio-temporal data by proposing a parallel and scalable solution. Thus, we turn our attention to large-scale trajectory data sets that exceed the storage and processing capabilities of a single centralized node. We assume that the data set $\mathcal{D}$ is stored distributed in multiple nodes, without any more specific assumptions about the exact partitioning mechanism. Put differently, a node stores a subset D_i of the records of $\mathcal{D}$, and it holds that $D_i \bigcap D_j = \emptyset$ (for $i \neq j$), and $\bigcup D_i = \mathcal{D}$. Hence, in this paper, we study a distributed version of Problem 1. Table I provides an overview of the notation used in this paper.

III. Exact THS Algorithm

In this section, we present the THS (Trajectory Hot Spot) algorithm for distributed hot spot analysis over big trajectory data. The proposed algorithm is designed to be efficiently executed over a set of nodes in parallel and is implemented in Apache Spark. The input data set $\mathcal{D}$ is assumed to be stored in a distributed file system, in particular HDFS.

A. Overview

Intuitively, our solution consists of three main steps, which are depicted in Figure 1. In the first step, the goal is to compute all the cells' attribute values of a user defined spatio-temporal equi-width grid. To this end, the individual attribute values of trajectory data points are aggregated into cell attribute values, using the formula introduced in Section II. 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: $\sum_{j=1}^{n} w_{i,j}x_j$, by exchanging the cells' attribute values between themselves. 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 spatio-temporal grid cells, by applying the Getis-Ord formula. The trajectory hot spots can then be trivially calculated, by either selecting the top-k cells with the higher z-score values, or by selecting the cells having a p-value below a specified threshold.

The above description explains the rationale of our approach. In the following, we describe the implementation of our solution using Apache Spark Core API, along with the necessary technical details.

Algorithm 1 THS Step 1: Build Attribute Grid

```
1: Input: D, P
2: Output: gridRDD: RDD[i, x_i]
3: function
4: gridRDD = D.mapToPair(p =>
5:     emit new pair(getCellId(p) ⊕ p.τ, p.v)
6: ).reduceByKey(t_1, t_2 =>
7:     emit new pair(MIN(t_1, t_2), MAX(t_1, t_2))
8: ).mapToPair(cell_trajectory_id, (t_start , t_end) =>
9:     emit new pair(cell_id, (t_end − t_start) / (c_i_end − c_i_start))
10: ).reduceByKey(v_1, v_2 => emit v_1 + v_2)
11: end function
```

B. Building the Attribute Grid

The first step of THS, depicted in Algorithm 1, takes as input a data set $\mathcal{D}$ of trajectories stored in HDFS and a spatio-temporal partitioning $\mathcal{P}$, which defines the size of all cells c_i regarding their spatial and temporal dimensions. We use a function getCellId(p) to get the identifier i of cell c_i enclosing data point p. Initially, the trajectory data points are mapped to key value pairs (line 5), where the key is composed by a string concatenation (denoted with $\oplus$ in the algorithm) of the data point's cell id and its trajectory id ($p.\tau$), while the value is the timestamp of p. This assignment of composite keys, enables us to group data points by cell id and trajectory id; we calculate the minimum and maximum attribute values (t_{start}, t_{end} respectively) for each such group (line 7). Then, we compute the fraction $\frac{t_{end}-t_{start}}{c_{i_{end}}-c_{i_{start}}}$, individually for each group defined by the composite keys, and keep only the cell id part of the key (line 9). We perform a regrouping based on the new keys, to calculate the sum of the fractions for each cell (line 10). Hence, we have now successfully built the attribute grid (*gridRDD*), by computing each cell's attribute value $x_i = \sum_{t \in c_i} \frac{t_{end}-t_{start}}{c_{i_{end}}-c_{i_{start}}}$.

C. Exchanging Cell Information

In its second step, as presented in Algorithm 2, THS takes as input the attribute grid produced by the first step and the spatio-temporal partitioning $\mathcal{P}$. In line 4, we distributively calculate the sum and squared sum of the cells' attribute values. Having computed these sums, we can trivially calculate the mean value $\overline{X}$ and standard deviation S, in a centralized fashion (line 5).

Then, our goal is to broadcast each cell's weighted attribute value to all the other cells of the grid. To this end, in lines 7-10, we first get the list of weights between current cell c_i and

Algorithm 2 THS Step 2: Exchange Cell Information

1: **Input:** gridRDD: RDD[i, x_i], $\mathcal{P}$
2: **Output:** $\overline{X}$, S, wSumRDD: RDD[i, $\sum_{j=1}^{n} w_{i,j}x_j$]
3: **function**
4: gridRDD.forEach(x_i => update accumulators)
5: calculate $\overline{X}$ and S from accumulators
6: wSumRDD = gridRDD.flatMapToPair(i, x_i =>
7: w_i = getWeightList(i)
8: **for each** j **in** w_i **do**
9: emit **new** pair(j, $x_i * w_{i,j}$)
10: **end for**
11:).reduceByKey(wx_1, wx_2 => emit $wx_1 + wx_2$)
12: **end function**

all the cells c_j of the grid (line 7). For each cell c_j we emit a new key value pair consisting of the j value as the key and the weighted attribute as its value (line 9). Then, we perform a grouping of these key value pairs, based on their keys (i.e., cell ids), while calculating the sum of their weighted attribute values (line 11). The result of this operation is the weighted sum grid (*wSumRDD*), which will be used for computing the Getis-Ord formula.

Algorithm 3 THS Step 3: Calculate Getis-Ord

1: **Input:** $\overline{X}$, S, wSumRDD: RDD[i, $\sum_{j=1}^{n} w_{i,j}x_j$], $\mathcal{P}$
2: **Output:** G$_i$RDD: RDD[i, G_i^*]
3: **function**
4: G$_i$RDD = wSumRDD.mapToPair(i, wx_i =>
5: sum_i = getWeightSum(i)
6: $squaredSum_i$ = getWeightSquaredSum(i)
7: emit **new** pair(i, $\frac{wx_i - \overline{X} \cdot sum_i}{S\sqrt{\frac{[n \cdot squaredSum_i - (sum_i)^2]}{n-1}}}$)
8:)
9: **end function**

D. Calculating z-scores with Getis-Ord statistic

The third step of THS is depicted in Algorithm 3. It takes as input the $\overline{X}$ and S values computed in the previous part, along with the weighted sum grid and the spatio-temporal partitioning $\mathcal{P}$. We map each cell's weighted sum attribute value to a z-score by applying the Getis-Ord formula. The sum and squared sum of weights are initially computed (lines 5,6) in order to be provided to the calculation of the Getis-Ord formula right after (line 7). Finally, the result of this operation, results to a data set (*G$_i$RDD*) consisting of cell ids and their Getis-Ord z-scores.

IV. An Approximate Algorithm: aTHS

The afore-described algorithm (THS) is exact and computes the correct hot spots over widely distributed data. However, its computational cost is relatively high and can be intolerable when the number of cells n in $\mathcal{P}$ is large. This is because every cell's 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.

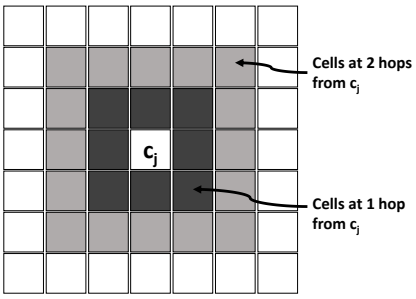


Fig. 2. Example of cells at distance from a reference cell c_j (the dark color indicates the weight of their contribution to c_j's value x_j).

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 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 hot spot 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.

A. 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 Figure 2, 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. Algorithm 4 describes the pseudo-code of the second step of aTHS. The main difference is in line 9, where we check the distance between cells i and j; if the distance is greater than the threshold h, then the emission of a new key value pair does not occur (i.e., we do not broadcast the weighted attribute value of cell c_i to cell c_j).

The third step of aTHS calculates the weight attribute sum and squared sum, by applying a weight factor equal to zero, for cells c_j which are located in a distance farther than h from c_i. This change affects only the implementation of the

Algorithm 4 aTHS Part 2: Exchange Cell Information

1: **Input:** gridRDD: RDD[i, x_i], $\mathcal{P}$, h
2: **Output:** $\overline{X}$, S, wSumRDD: RDD[i, $\sum_{j=1}^{n} w_{i,j}x_j$]
3: **function**
4: gridRDD.forEach(x_i => update accumulators)
5: calculate $\overline{X}$ and S from accumulators
6: wSumRDD = gridRDD.flatMapToPair(i, x_i =>
7: w_i = getWeightList(i)
8: **for each** j **in** w_i **do**
9: **if** DISTANCE(i, j) $\leq h$ **then**
10: emit **new** pair(j, $x_i * w_{i,j}$)
11: **end if**
12: **end for**
13:).reduceByKey(wx_1, wx_2 => emit $wx_1 + wx_2$)
14: **end function**

getWeightSum and *getWeightSquaredSum* methods, used in lines 5,6 of Algorithm 3.

B. Controlling the Error ΔG_i^*

In this section, we provide an upper bound for the value ΔG_i^*. We use $\mathcal{H}_{ij}$ to denote the distance measured in number of cells between cells c_i and c_j. For example, in Figure 2, the darker colored cells are 1 hop away, while the lighter grey colored cells are 2 hops away from the reference cell c_j. Thus, we have:

$$\Delta G_i^* = G_i^* - \hat{G}_i^* =$$

$$\frac{\sum_{j=1}^{n} w_{i,j}x_j - \overline{X}\sum_{j=1}^{n} w_{i,j}}{S\sqrt{\frac{[n\sum_{j=1}^{n} w_{i,j}^2 - (\sum_{j=1}^{n} w_{i,j})^2]}{n-1}}} - \frac{\sum_{\mathcal{H}_{ij}\leq h} w_{i,j}x_j - \overline{X}\sum_{\mathcal{H}_{ij}\leq h} w_{i,j}}{S\sqrt{\frac{[n\sum_{\mathcal{H}_{ij}\leq h} w_{i,j}^2 - (\sum_{\mathcal{H}_{ij}\leq h} w_{i,j})^2]}{n-1}}}$$

Since the expression $n\sum_{j=1}^{n} w_{i,j}^2 - (\sum_{j=1}^{n} w_{i,j})^2$ is positive monotone with increasing n values, we can write the following inequality by replacing the denominator of the first fraction with the second:

$$\Delta G_i^* \leq$$

$$\frac{\sum_{j=1}^{n} w_{i,j}x_j - \overline{X}\sum_{j=1}^{n} w_{i,j} - (\sum_{\mathcal{H}_{ij}\leq h} w_{i,j}x_j - \overline{X}\sum_{\mathcal{H}_{ij}\leq h} w_{i,j})}{S\sqrt{\frac{[n\sum_{\mathcal{H}_{ij}\leq h} w_{i,j}^2 - (\sum_{\mathcal{H}_{ij}\leq h} w_{i,j})^2]}{n-1}}}$$

To simplify notation, let us refer to the denominator of the above expression as: $\gamma = S\sqrt{\frac{[n\sum_{\mathcal{H}_{ij}\leq h} w_{i,j}^2 - (\sum_{\mathcal{H}_{ij}\leq h} w_{i,j})^2]}{n-1}}$, then the error ΔG_i^*:

$$\Delta G_i^* \leq$$

$$\frac{1}{\gamma}[\sum_{j=1}^{n} w_{i,j}x_j - \overline{X}\sum_{j=1}^{n} w_{i,j} - (\sum_{\mathcal{H}_{ij}\leq h} w_{i,j}x_j - \overline{X}\sum_{\mathcal{H}_{ij}\leq h} w_{i,j})] =$$

$$\frac{1}{\gamma}(\sum_{\mathcal{H}_{ij}>h} w_{i,j}x_j - \overline{X}\sum_{\mathcal{H}_{ij}>h} w_{i,j})$$

Further, let x_{max} denote the maximum value of x_i for any cell c_i in the grid. Then, we can replace x_j with x_{max} and we have:

$$\Delta G_i^* \leq \frac{x_{max} - \overline{X}}{\gamma}\sum_{\mathcal{H}_{ij}>h} w_{i,j}$$

It is trivial to derive a formula that, for a given cell, returns the number of cells at distance ρ. Function: $f(\rho, d)$ returns the *total* number of cells (including the given cell) within ρ hops from the given cell: $f(\rho, d) = (2\rho + 1)^d$, where d is the dimensionality. For example, for ρ=1 and d=3, $f(1, 3) = 3^3 = 27$, as expected. Then, the formula[2] returning the number of cells at exactly ρ hops away from a given cell is: $f(\rho, d) - f(\rho - 1, d)$.

Also, recall that by definition $w_{ij} = a^{1-\rho}$ for cells c_i and c_j which are located at ρ hops away, then:

$$\sum_{\mathcal{H}_{ij}>h} w_{i,j} = \sum_{\rho>h} a^{1-\rho}[f(\rho, d) - f(\rho - 1, d)]$$

Putting everything together:

$$\Delta G_i^* \leq \frac{x_{max} - \overline{X}}{\gamma}\sum_{\rho>h} a^{1-\rho}[f(\rho, d) - f(\rho - 1, d)]$$

In summary, we can compute an upper bound for the error ΔG_i^* introduced to the Getis-Ord value of a cell c_i, due to approximate processing using only neighbors at distance h. In turn, this allows us to explicitly set the value h in Algorithm aTHS, in such a way that we can guarantee that the maximum error introduced is quantified. In practice, an analyst can exploit our method to trade-off accuracy for computational efficiency, making aTHS an attractive algorithm for trajectory hot spot analysis over massive data sets.

V. Experimental Evaluation

In this section, we evaluate the performance of our approach for trajectory hot spot analysis. We implemented all algorithms in Java, using Apache Spark 2.2.0 Core API. Our Java code can be found at: https://github.com/nikpanos/trajectory_hotspot_analysis/tree/IEEE_BigData18.

[2]We acknowledge that function $f(\rho, d)$ provides an overestimation of the number of cells, when the cell under study is situated at distance smaller than ρ from the grid boundaries, so we use this expression as an upper bound.

A. Experimental Setup

Platform. We deployed our code on a Hadoop YARN 2.7 cluster consisting of 10 computing nodes. In all our experiments, we use the YARN cluster deploy mode. We initiate 9 spark executors, configured to use 5.5GB of main memory and 2 executor cores each. We also configured HDFS with 128MB block size and a replication factor of 2.

Data sets. We employed a real data set containing surveillance information from the maritime domain. The data was collected over a period of three years, consisting of 83,735,633 individual trajectories for vessels moving in the Eastern Mediterranean Sea. This data set is 89.4GB 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 $\langle trajectoryID; timestamp; latitude; longitude\rangle$. The data set is stored in 720 HDFS blocks, in uncompressed text format.

Selecting hot spots. In order to identify hot spots on trajectory data, we opt to select them by reporting in the final result set only the top-k grid cells having the highest Getis-Ord z-scores. To this end, after the execution of the third step of our solution, we perform a distributed sorting of all the cells in *G_iRDD* data set by descending order of their z-scores, and then, we pick the first k cells. Hence, our experimental study includes a fourth step, which concerns the discovery of top-k cells. The implementation of this fourth step is straightforward, as the Spark Core API provides all the necessary methods for sorting an RDD, and then collecting the first k values from it.

Metrics. Our main evaluation metric was the execution time needed for each individual step of our algorithm to complete in our experiments 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.

Furthermore, in each experiment, we measured (a) the total number n of cells in $\mathcal{P}$, (b) the size $|$*gridRDD*$|$ of non-empty cells in $\mathcal{P}$ and (c) the size $|$*wSumRDD*$|$ of weighted attribute sum data, after executing the second step of aTHS (Algorithm 4). Essentially, these values greatly affect the performance of our solution, especially regarding the size of produced network traffic, thus providing a deeper insight to the complexity of our algorithm. Notice that by definition $|$*gridRDD*$| \leq |$*wSumRDD*$| \leq n$, since $|$*gridRDD*$|$ depends on the data distribution, and $|$*wSumRDD*$|$ depends on the value h.

Notice that we use log-scale on the y axis in all our charts, throughout the experimental section.

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 hot spots to be reported in the final result set. In practice, the first two parameters affect the number n of cells of $\mathcal{P}$, 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.

Parameter	**Values**
Spatial cell size of $\mathcal{P}$ (degrees)	0.05, **0.1**, 0.5
Temporal cell size of $\mathcal{P}$ (hours)	1, **2**, 12
h	1, **2**, 3
top-k	50, **100**, 500

TABLE II
EXPERIMENTAL SETUP PARAMETERS (DEFAULT VALUES IN BOLD).

For convenience, we briefly recall the steps of our algorithms here:

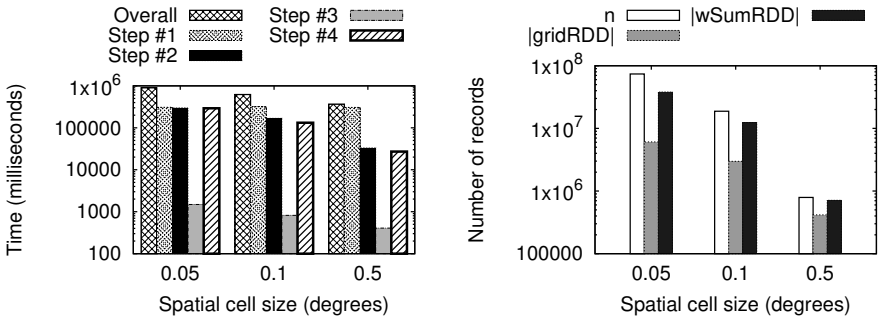
- Step #1 (Building the Attribute Grid): The input data set of trajectories is transformed and aggregated to a set of cells (*gridRDD*), where each cell c_i is expressed by its id i and attribute value x_i.
- Step #2 (Exchanging Cell Information): First we compute the mean value $\overline{X}$ and standard deviation S of the attribute values x_i. Then, each cell of *gridRDD* broadcasts its weighted attribute value to its neighbors and computes its weighted attribute sum. The resulting data set (*wSumRDD*) consists of cells c_i which are expressed by their id i and weighted attribute sum $\sum_{j=1}^{n} w_{i,j}x_j$.
- Step #3 (Calculating z-scores): For each cell in *wSumRDD* we calculate its Getis-Ord z-score. The resulting RDD consists of cells c_i which are expressed by their id i and z-score G_i^*.
- Step #4 (Sorting and reporting top-k): The result is sorted by z-scores in descending order and we output only the first k cells having the higher z-scores.

Our experimental setup is summarized in Table II. In each experiment, we vary a single parameter, while setting the others to their default values.

B. Experimental Results

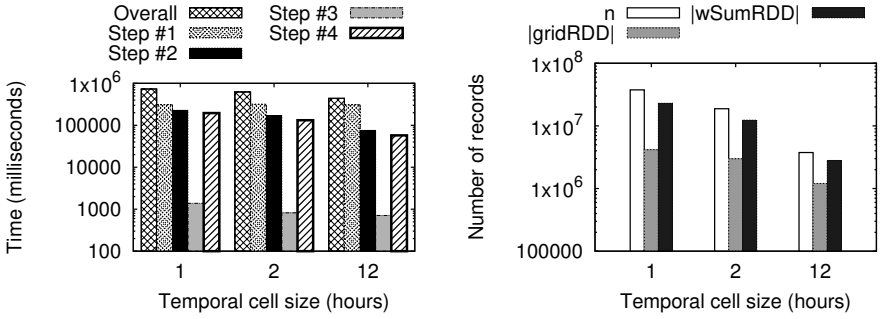
Varying the spatial cell size. In Figure 3, we demonstrate the results of our experiments, by varying the spatial cell size of $\mathcal{P}$. Higher sized spatial cells decrease the total number of cells (n) used in the grid partitioning of the 3D space. In turn, this is expected to lead to reduced execution 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.

In terms of individual steps, the first step of our solution, mostly depends on the size of input data set, thus it is not significantly affected by the spatial size of cells, as shown in Figure 3a. On the other hand, by using a larger cell size, the performance of the remaining steps is improved, as the data is aggregated into smaller number of cells. Put differently, all constructed RDDs contain fewer objects and are of smaller size, thus can be processed faster and produce less network overhead by data exchange. Specifically, the execution time



(a) Execution Time (b) Number of Cells

Fig. 3. Performance of our algorithm for various spatial cell sizes of $\mathcal{P}$.



(a) Execution Time (b) Number of Cells

Fig. 4. Performance of our algorithm for various temporal cell sizes of $\mathcal{P}$.

of steps two and four follows linearly the spatial size of cells, since for ten times larger cells, the execution time is ten times less. Furthermore, step three proves to be the most efficient, justified by the fact that it does not include any operation involving the network.

Figure 3b depicts the total number of cells (n) in $\mathcal{P}$, along with the number of cells contained in *gridRDD* and *wSumRDD*, when using various sizes of cells for their spatial dimensions. As expected, the total number of cells decreases for a more coarse spatial partitioning. Additionally, following the input data distribution, the relative number of empty cells in the grid increases when using a finer partitioning scheme, thus affecting the size of *gridRDD* data set. However, these empty cells may have a non-zero weighted sum attribute value, if they are located closely to other non-empty cells. This is demonstrated by the fact that *wSumRDD* is larger than *gridRDD*, since the former includes also some empty cells which have a non-zero weighted sum attribute value.

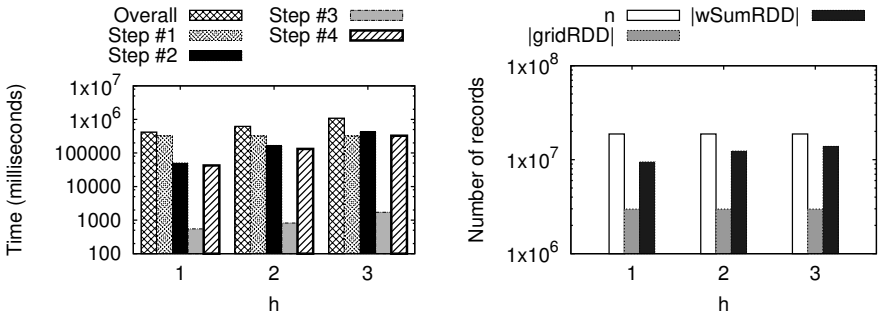
Varying the temporal cell size. Figure 4 demonstrates the efficiency of our algorithm for various temporal cell sizes. The effect of larger cells in the temporal dimension to the overall execution time is similar to the previous experiment: larger temporal cell size leads to fewer total cells in the grid, thus to reduced overall execution time. The experiment with the most coarse temporal partitioning (12 hours), was measured to be twice more efficient than the experiment using the finest partitioning, in total execution time.

In terms of individual steps, the first step is (again) not significantly affected by the size of the cell, since its associated cost is dominated by the overhead of reading data from the disk. The execution time of the rest of the steps, appears to be higher for a finer temporal partitioning scheme, as depicted in Figure 4a.

Similarly, the number of total cells in $\mathcal{P}$ decreases for a higher value of temporal cell size, as depicted in Figure 4b. The number of empty cells naturally increases for a finer defined grid, resulting to smaller *gridRDD* and *wSumRDD* data sets.

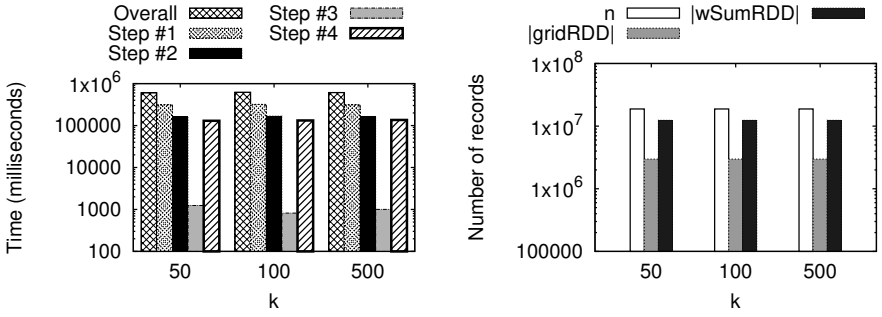
Varying h. aTHS can be parameterized with a user-defined variable h, which defines the set of neighboring cells contributing to the calculation of each cell's z-score. Figure 5 demonstrates the experimental results when using different values for variable h. The overall execution time is significantly reduced for lower values of h, since each cell broadcasts its attribute value to fewer 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 to an approximate result $\hat{G}_i^*$. However, the deviation of $\hat{G}_i^*$ to G_i^* can be quantified as explained in Section IV-B.

The execution time of our algorithm's first step, is not affected by the value of h, since it is not dependent on this



(a) Execution Time (b) Number of Cells

Fig. 5. Performance of our algorithm for various values of h.



(a) Execution Time (b) Number of Cells

Fig. 6. Performance of our algorithm for various values of k.

variable. As such, the size of *gridRDD* data set is also not affected by the value of h, as shown in Figure 5b.

By using a larger value of h, the size of *wSumRDD* increases, due to the fact that more empty cells obtain a weighted attribute sum which is greater than zero. This data set is produced during the second step of our solution, while steps three and four operate on data sets having the same size as *wSumRDD*. Hence, larger values of h result to higher execution times for these individual steps.

Varying top-k. The value of top-k affects the size of the final result set. Figure 6 demonstrates the impact of this variable to data set 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. As shown in Figure 6a, the first three steps are not affected by the value of this variable, as their goal is to compute the z-scores of all cells in $\mathcal{P}$. This fact is also reflected in Figure 6b, where the size of the data sets is the same for all values of k. The fourth step of our solution requires slightly higher processing time to produce larger result set size, but the increase of its relative processing cost is negligible.

VI. Related Work

Hot spot analysis is the process of identifying statistically significant clusters. This kind of analysis is often confused with clustering with respect to the density of the identified groups [12]. The computation of density gives us information where clusters in our data exist, but not whether the clusters are statistically significant; that is the outcome of hot spot analysis. In geospatial analysis the hot spot discovery is performed with spatial statistics like Moran's I [13] or Getis-ord Gi* [11] that measure spatial autocorrelation based on feature locations and attribute values, while they result in z-scores and p-values for the predetermined points or regions of interest. A high z-score and small p-value indicates a significant hot spot. A low negative z-score and small p-value indicates a significant cold spot. The higher (or lower) the z-score, the more intense the clustering. A z-score near zero means no spatial clustering [11].

Trajectory hot spot analysis is related to the problem of finding interesting places. In [14], interesting places are defined as either: (a) the areas where a single moving object spends a large amount of time, or (b) the frequently visited areas from various trajectories, or (c) the areas where many trajectories alter their state or behavior. In [15] interesting places are identified as areas where several moving objects spend large amount of time, by moving with low speed. The minimum amount of moving objects and their minimum duration of stay, should be provided by the user at query time. To enable efficient execution for various parameters, an indexing structure is proposed which enables fast retrieval of trajectory segments based on their speeds. Notice that these variations aim to discover spatial regions of interest, while

our approach identifies interesting spatial regions for various temporal segments.

Hot spot analysis is a special case of spatio-temporal data analysis, mobility data mining and more specifically trajectory data mining, since we are interested in trajectory-based hot spot analysis. These domains have been the subject of many research efforts lately. Recent works on hot spot analysis for spatio-temporal data include [16], [17]. The study in [17] proposes a different way to visualize and analyze spatio-temporal data. The aim is to identify areas of high event density, by using multivariate kernel density estimation. Different kernels in spatial and temporal domains can be used. After such hot spots have been identified, a spatio-temporal graph can be formed to represent topological relations between hot spots. In [16], a spatio-temporal graph is analyzed in order to find anomalies on people's regular travel patterns. These interesting phenomena are called black holes and volcanos, represented as subgraphs with overall inflow traffic greater than the overall outflow by a threshold and vice-versa. The detection of frequent patterns and relations between black holes and volcanos lead to the discovery of human mobility patterns. In [18], hot spot analysis is used for studying mobile traffic. The aim is to identify locations where the density of data volumes transmitted is high, based on specific values of thresholds. The results of the analysis are then used to detect the distribution of mobile data traffic hot spots and to propose a meaningful cell deployment strategy.

In the domain of trajectory data mining [19] there are several clustering approaches that are relevant to this work. The typical approach is to either transform trajectories to vector data, in order for well-known clustering algorithms to be applicable, or to define appropriate trajectory similarity functions, which is the basic building block of every clustering approach. For instance, CenTR-I-FCM [20] builds upon a Fuzzy C-Means variant to perform a kind of time-focused local clustering using a region growing technique under similarity and density constraints. For each time period, the algorithm determines an initial seed region (that corresponds to the sub-trajectory restricted inside the period) and searches for the maximum region that is composed of all sub-trajectories that are similar with respect to a distance threshold d and dense with respect to a density threshold . Subsequently, the growing process begins and the algorithm tries to find the next region to extend among the most similar sub-trajectories. The algorithm continues until no more growing can be applied, appending in each repetition the temporally local centroid. In the same line of research, having defined an effective similarity metric, TOPTICS [21] adapts OPTICS [22] to enable whole-trajectory clustering (i.e., clustering the entire trajectories), TRACLUS [23] exploits on DBSCAN [24] to support sub-trajectory clustering, while T-Sampling [25], [26], introduces trajectory segmentation (aiming at temporal-constrained sub-trajectory clustering [27]), by taking into account the neighborhood of a trajectory in the rest of the data set, yielding a segmentation that is related only on the number of neighboring segments that vote for the line segments of a trajectory as the most representatives. All the above trajectory clustering approaches they are capable of identifying trajectory clusters and their densities but do not tackle the issue of statistical significance in the space-time they take place.

There are several other methods that try to identify frequent (thus dense) trajectory patterns. In case where moving objects move under the restrictions of a transportation network, [28] proposed an online approach to discover and maintain hot motions paths while [29] tackled the problem of discovering the most popular route between two locations based on the historical behavior of travelers. In case where objects move without constraints, [30] proposed a method to discover collocation patterns, while in [31] where the goal is to discover sequential trajectory patterns (T-patterns), the popular regions that participate in T-patterns can be computed automatically following a clustering approach that utilizes the density of the trajectories in the space. The algorithm starts by tessellating the space into small rectangular cells, for each of which the density (i.e., the number of trajectories that either cross it or found inside the cell) is computed. Then, by following a region-growing technique the dense areas are enlarged by including nearby cells as long as the density criterion is fulfilled. The problem of identifying hot spots from trajectory data in indoor spaces has been studied in [32]. It introduces a formula for computing scores for indoor locations based on users' interests rather than measuring the amount of time a user spends in a specific area. This formula is used for calculating a score for each indoor area, based on the mutual reinforcement relationship between users and indoor locations. These methods have a different focus from our proposal sharing similar objectives and shortcomings as the aforementioned techniques.

The problem of finding hot spots for spatio-temporal point data has been studied in SigSpatial Cup at 2016 (http://sigspatial2016.sigspatial.org/giscup2016/home) where several interesting methods where proposed. Among others, [33], [34] proposed algorithms to identify hot spots based on spatial density of point data (in particular drop-off locations of taxis). Instead, in this paper, we study the problem of hot spot analysis for trajectory data, which is different because the effect of a data point to a cell depends on the trajectory in which it belongs to (i.e., on other points). To the best of our knowledge, there is a lack of parallel processing solutions that operate on distributed trajectory data in an efficient and scalable way to discover trajectory-based hot spots.

VII. Conclusions

In this paper, we formulate the problem of Trajectory hot spot analysis, which finds many real-life applications in the case of big trajectory data. We propose two parallel data processing algorithms that solve the problem, which are scalable for data of large volumes. Our first algorithm (THS) provides an exact solution to the problem, but may be computationally expensive depending on the underlying grid partitioning and the number of cells. Also, we propose an approximate algorithm (aTHS) that practically ignores the contribution of cells located further away from the cell

under study, thereby saving computational cost. Perhaps more importantly, we propose a method that can be used to bound the error in the approximation, for a given subset of cells that are taken into account. Thus, we can trade-off accuracy for efficiency in a controlled manner. Our implementation is based on Apache Spark, and we demonstrate the scalability of our approach using a data set of vessel trajectories that spans three years in total. In our future work, we intend to explore variants of the problem of trajectory hot spot analysis, which is a problem that deserves further study, also for data sets coming from other domains, such as the aviation or urban domain.

VIII. Acknowledgments

This research work has received funding from the Hellenic Foundation for Research and Innovation (HFRI) and the General Secretariat for Research and Technology (GSRT), under the HFRI PhD Fellowship grant (GA. no. 1059). This work was partially supported by projects datACRON (grant agreement No 687591), Track&Know (grant agreement No 780754) and MASTER (Marie Sklowdoska-Curie agreement N. 777695), which have received funding from the EU Horizon 2020 R&I Programme.

References

[1] A. Eldawy and M. F. Mokbel, "The era of big spatial data: A survey," *Foundations and Trends in Databases*, vol. 6, no. 3-4, pp. 163–273, 2016.

[2] L. Alarabi and M. F. Mokbel, "A demonstration of st-hadoop: A mapreduce framework for big spatio-temporal data," *PVLDB*, vol. 10, no. 12, pp. 1961–1964, 2017.

[3] L. Alarabi, M. F. Mokbel, and M. Musleh, "St-hadoop: A mapreduce framework for spatio-temporal data," in *Proceedings of the 15th International Symposium on Spatial and Temporal Databases, SSTD*, 2017, pp. 84–104.

[4] S. Hagedorn and T. Räth, "Efficient spatio-temporal event processing with STARK," in *Proceedings of the 20th International Conference on Extending Database Technology, EDBT*, 2017, pp. 570–573.

[5] M. Tang, Y. Yu, Q. M. Malluhi, M. Ouzzani, and W. G. Aref, "Locationspark: A distributed in-memory data management system for big spatial data," *PVLDB*, vol. 9, no. 13, pp. 1565–1568, 2016.

[6] C. Doulkeridis, A. Vlachou, D. Mpestas, and N. Mamoulis, "Parallel and distributed processing of spatial preference queries using keywords," in *Proceedings of the 20th International Conference on Extending Database Technology, EDBT*, 2017, pp. 318–329.

[7] R. T. Whitman, M. B. Park, B. G. Marsh, and E. G. Hoel, "Spatio-temporal join on apache spark," in *Proceedings of the 25th ACM International Conference on Advances in Geographic Information Systems, SIGSPATIAL*, 2017, pp. 20:1–20:10.

[8] Y. Xian, Y. Liu, and C. Xu, "Parallel gathering discovery over big trajectory data," in *Proceedings of the 2016 IEEE International Conference on Big Data, BigData*, 2016, pp. 783–792.

[9] Y. Fang, R. Cheng, W. Tang, S. Maniu, and X. S. Yang, "Scalable algorithms for nearest-neighbor joins on big trajectory data," *IEEE Transactions on Knowledge and Data Engineering TKDE*, vol. 28, no. 3, pp. 785–800, 2016.

[10] C. Claramunt, C. Ray, E. Camossi, A. Jousselme, M. Hadzagic, G. L. Andrienko, N. V. Andrienko, Y. Theodoridis, G. A. Vouros, and L. Salmon, "Maritime data integration and analysis: recent progress and research challenges," in *Proceedings of the 20th International Conference on Extending Database Technology, EDBT*, 2017, pp. 192–197.

[11] J. K. Ord and A. Getis, "Local spatial autocorrelation statistics: Distributional issues and an application," *Geographical Analysis*, vol. 27, no. 4, pp. 286–306, October 1995.

[12] P. Zhao, K. Qin, Q. Zhou, C. Liu, and Y. Chen, "Detecting hotspots from taxi trajectory data using spatial cluster analysis," *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 2, no. 4, pp. 131–135, 2015.

[13] P. Moran, "Notes on continuous stochastic phenomena," *Biometrika*, vol. 37, no. 1, pp. 17–23, 1950.

[14] J. Gudmundsson, M. J. van Kreveld, and F. Staals, "Algorithms for hotspot computation on trajectory data," in *Proceedings of the 21st International Conference on Advances in Geographic Information Systems, SIGSPATIAL*, 2013, pp. 134–143.

[15] M. R. Uddin, C. Ravishankar, and V. J. Tsotras, "Finding regions of interest from trajectory data," in *Proceedings of the 12th International Conference on Mobile Data Management MDM*, vol. 1, 2011, pp. 39–48.

[16] L. Hong, Y. Zheng, D. Yung, J. Shang, and L. Zou, "Detecting urban black holes based on human mobility data," in *Proceedings of the 23rd International Conference on Advances in Geographic Information Systems SIGSPATIAL*, 2015, pp. 35:1–35:10.

[17] J. Lukasczyk, R. Maciejewski, C. Garth, and H. Hagen, "Understanding hotspots: A topological visual analytics approach," in *Proceedings of the 23rd International Conference on Advances in Geographic Information Systems SIGSPATIAL*, 2015, pp. 36:1–36:10.

[18] H. Klessig, V. Suryaprakash, O. Blume, A. J. Fehske, and G. Fettweis, "A framework enabling spatial analysis of mobile traffic hot spots," *IEEE Wireless Commun. Letters*, vol. 3, no. 5, pp. 537–540, 2014.

[19] Y. Zheng, "Trajectory data mining: An overview," *ACM Transactions on Intelligent Systems and Technology TIST*, vol. 6, no. 3, pp. 29:1–29:41, 2015.

[20] N. Pelekis, I. Kopanakis, E. E. Kotsifakos, E. Frentzos, and Y. Theodoridis, "Clustering uncertain trajectories," *Knowledge and Information Systems*, vol. 28, no. 1, pp. 117–147, 2011.

[21] M. Nanni and D. Pedreschi, "Time-focused clustering of trajectories of moving objects," *J. Intell. Inf. Syst.*, vol. 27, no. 3, pp. 267–289, 2006.

[22] M. Ankerst, M. M. Breunig, H. Kriegel, and J. Sander, "OPTICS: ordering points to identify the clustering structure," in *Proceedings of the 1999 ACM International Conference on Management of Data, SIGMOD*, 1999, pp. 49–60.

[23] J. Lee, J. Han, and K. Whang, "Trajectory clustering: a partition-and-group framework," in *Proceedings of the ACM International Conference on Management of Data, SIGMOD*, 2007, pp. 593–604.

[24] M. Ester, H. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining KDD*, 1996, pp. 226–231.

[25] N. Pelekis, I. Kopanakis, C. Panagiotakis, and Y. Theodoridis, "Unsupervised trajectory sampling," in *Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases, ECML PKDD, Part III*, 2010, pp. 17–33.

[26] C. Panagiotakis, N. Pelekis, I. Kopanakis, E. Ramasso, and Y. Theodoridis, "Segmentation and sampling of moving object trajectories based on representativeness," *IEEE Transactions on Knowledge and Data Engineering TKDE*, vol. 24, no. 7, pp. 1328–1343, 2012.

[27] N. Pelekis, P. Tampakis, M. Vodas, C. Doulkeridis, and Y. Theodoridis, "On temporal-constrained sub-trajectory cluster analysis," *Data Mining and Knowledge Discovery*, vol. 31, no. 5, pp. 1294–1330, 2017.

[28] D. Sacharidis, K. Patroumpas, M. Terrovitis, V. Kantere, M. Potamias, K. Mouratidis, and T. K. Sellis, "On-line discovery of hot motion paths," in *Proceedings of the 11th International Conference on Extending Database Technology, EDBT*, 2008, pp. 392–403.

[29] Z. Chen, H. T. Shen, and X. Zhou, "Discovering popular routes from trajectories," in *Proceedings of the 27th International Conference on Data Engineering, ICDE*, 2011, pp. 900–911.

[30] H. Cao, N. Mamoulis, and D. W. Cheung, "Discovery of collocation episodes in spatiotemporal data," in *Proceedings of the 6th IEEE International Conference on Data Mining, ICDM*, 2006, pp. 823–827.

[31] F. Giannotti, M. Nanni, F. Pinelli, and D. Pedreschi, "Trajectory pattern mining," in *Proceedings of the 13th ACM International Conference on Knowledge Discovery and Data Mining, SIGKDD*, 2007, pp. 330–339.

[32] P. Jin, J. Du, C. Huang, S. Wan, and L. Yue, "Detecting hotspots from trajectory data in indoor spaces," in *Proceedings of the 20th International Conference on Database Systems for Advanced Applications, DASFAA, Part I*, 2015, pp. 209–225.

[33] G. Makrai, "Efficient method for large-scale spatio-temporal hotspot analysis," in *Proceedings of the 24th ACM International Conference on Advances in Geographic Information Systems, SIGSPATIAL*, 2016.

[34] P. Nikitopoulos, A.-I. Paraskevopoulos, C. Doulkeridis, N. Pelekis, and Y. Theodoridis, "BigCAB: Distributed hot spot analysis over big spatio-temporal data using Apache Spark," in *Proceedings of the 24th ACM International Conference on Advances in Geographic Information Systems, SIGSPATIAL*, 2016.