Hot Spot Analysis for Big Trajectory Data in Road Networks

Panagiota Keziou, Christos Doulkeridis

Department of Digital Systems, University of Piraeus, Greece

Abstract

Hot spot analysis is the problem of identifying statistically significant spatial clusters and is typically applied on point data. The aim of this paper is to discover hot spots in an urban environment, namely road segments with statistically significant amount of traffic congestion, using massive GPS data of moving objects. To this end, we adjust the Getis-Ord index for hot spot discovery to become applicable for road networks. Then, we propose two data-parallel algorithms for hot spot discovery in road networks implemented in Apache Spark; an exact algorithm that has scalability limitations for very large data sets, and a scalable approximate algorithm that balances between performance and accuracy. We provide experiments over a large real-life data set that indicate the salient features of our approach.

Keywords

Hot spot analysis, road networks, big data, urban mobility

1. Introduction

In recent years, the analysis of mobility data has drawn the attention of both the academic society and the industry. Tracking of moving objects on road networks is ubiquitous in fleet management applications and can support diverse scenarios of urban planning, more efficient transportation, reduction of greenhouse gas emissions, and improvement of life quality. Due to the technological evolution of GPS sensors and their integration in mobile devices and cars, it is possible to collect vast amounts of spatio-temporal data. Thus, our work is motivated by the need for identifying traffic congestion from GPS data on the road network.

In this paper, we present an approach for identifying traffic hot spots, using big trajectory data collected by GPS sensors. Through hot spot analysis, we aim to identify road segments with *statistically significant* high levels of traffic congestion. This means that we do not just aim to identify road segments that have high congestion values; such road segments may be of interest, but they are not necessarily statistically significant. Instead, a hot spot is statistically significant if it has high congestion value but additionally is surrounded by other segments with high congestion values.

For this purpose, we employ the Getis-Ord Statistic index, a popular metric for hot spot analysis, which – to the best of our knowledge – has so far been used for spatial and spatio-temporal data of objects with free movement. Instead, we adapt the Getis-Ord index to become applicable for graphs that represent the road network, such as the ones obtainable via OpenStreetMap. Moreover, to handle the temporal dimension, we consider different temporal snapshots of the road network whose duration is user-defined according to the application requirements. Effectively, we have multiple snapshots of the same graph, each corresponding to the same road network but for a different time interval and thus each road segment has different traffic congestion values. In this way, we formulate the problem of hot spot discovery in road networks using a model that consists of a sequence of temporal snapshots of the same graph (road network).

Computing the hot spots using this model is a processing-intensive operation, which relies on diffusion of information over (parts of) the graphs. Moreover, the necessary data to infer traffic congestion on roads may be in the order of millions of GPS records per temporal interval. Therefore, we need to design scalable algorithms for hot spot analysis by exploiting big data technologies. To this end, we introduce two data-parallel algorithms for hot spot discovery in road networks, with a goal to be applicable for massive trajectory data sets. The first algorithm returns the exact result, but faces performance limitations for really large data sets due to the underlying exchange of information between pairs of edges. Then, we present an approximate algorithm with a cut-off threshold for excluding edges that are located too far away spatially or temporally, as their contribution to hot spot discovery is minimal. It turns out that our approximate algorithm provides an interesting trade-off between scalability and accuracy, returning hot spots that are very similar to ones discovered by the exact algorithm but orders of magnitude faster.

In summary, we make the following contributions:

- We formulate the problem of hot spot analysis in road networks, by the means of Getis-Ord Statistic, appropriately tailored for graphs.
- We design and implement an exact algorithm in Apache Spark that computes hot spots in parallel.
- We present a scalable approximate algorithm that discovers hot spots of high accuracy but significantly faster.
- We evaluate the performance and scalability of our algorithms for different parameters, using a real-life data set from the Metropolitan Area of Athens.

The rest of this paper is structured as follows: Section 2 reviews related research efforts. Section 3 formulates the problem under discussion. In Section 4, we present our approach for hot spot analysis over road networks, while in Section 5 we present the parallel algorithms for detecting hot spots. Then, in Section 6 describes the results of our evaluation, while in Section 7, we conclude the paper.

2. Related Work

Existing works in hot spot analysis over mobility data can be broadly classified in two categories: (a) point-based hot spot analysis, where mobility data consist of individual spatial or spatio-temporal points, and (b) trajectory-based hot spot analysis, where sequences of spatio-temporal points that belong to the same user/vehicle/vessel (i.e., trajectories) are considered. As our work belongs to the latter category, we

Published in the Proceedings of the Workshops of the EDBT/ICDT 2025 Joint Conference (March 25-28, 2025), Barcelona, Spain

keziou@hotmail.com (P. Keziou); cdoulk@unipi.gr (C. Doulkeridis)

briefly review point-based approaches and put our main focus on trajectory-based approaches.

2.1. Point-based Hot Spot Analysis

Eftelioglu et al. [1] propose the CGC (Cubic Grid Circle) algorithm for the problem of detecting geographically robust hot spots, modeled as circles in a 2D space. The proposed algorithm is an improvement of SaTScan algorithm, as it detects non-contiguous or sparse center hot spots and eliminates very small hot spots. In their studies, [2] and [3] expand the methodology for ring-shaped and elliptical hot spots detection, respectively. In [4], a density-based clustering algorithm (sig-DBSCAN) improves the quality of results by discarding random patterns, while the use of Dual-Convergence algorithm significantly reduces computational cost. For the above studies, the statistical significance of the proposed hot spots is assessed via Monte Carlo simulation. In contrast, Nikitopoulos et al. [5] propose a parallel and scalable solution for hot spot analysis over big spatio-temporal data. The spatio-temporal domain is divided into 3D cells and the Getis-Ord G_i^* statistic is used to detect the top-k statistically significant cells:

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

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 $\overline{X}$ and S represent the mean and standard deviation respectively:

$$\overline{X} = \frac{\sum_{j=1}^{n} x_j}{n}$$

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

In our work, we adapt the Getis-Ord statistic to be applicable on graphs representing temporal snapshots of road networks. The attribute x_j can be any indicator of traffic congestion, such as the average speed or the number of vehicles per time unit.

The proposed approaches of [6] and [7] take into account the underlying road network. Tang et al. [6] present a dynamic segmentation model of the road network, which allows the identification of statistically significant linear hot spots with high concentration of activities, such as pedestrian accidents on the road network, which otherwise would not have been recognized. According to the proposed approach, the shortest paths between activities are identified and a density index is calculated for each of them. The algorithm returns all the paths that are statistically significant (based on the Monte Carlo method) and their density value exceeds a threshold. In [7], the Network Isodistance Hotspot Detection (NIHD) problem is studied, which reveals sub-graph hot spots with high concentration of activities, diffused isotropically along the network. The deducted computational cost of the algorithm is achieved through network partitioning and upper-bound pruning.

2.2. Trajectory-based Hot Spot Analysis

In their studies, Häsner et al. [8] and Li et al. [9] take into account the limitations of the road network. Li et al. [9] propose the density-based algorithm FlowScan to discover patterns in the flow of traffic, referred to as "hot routes". By defining the minimum common traffic that successive road segments should share and the the number of edges that form a neighborhood, clusters of the moving objects are detected. The algorithm is capable of recognizing complex spatial patterns. The OPS methodology proposed by Häsner et al. [8] predicts near future traffic hot spots, based on recent positions of vehicles on the road network. A heuristic method is employed to predict the future position of the vehicles, and a weight is assigned to the nodes of the road network that reflects the likely traffic intensity within a specified time window. Finally, through an outlier detection approach, hot spots nodes are detected and the DBSCAN algorithm derives the sub-graphs that represent traffic hot spot regions.

Sacharidis et al. [10] propose a framework for on-line discovery of hot motion paths, in order to detect frequently travelled paths. Their study differentiates in the fact that the hotness of a road segment is determined as a function of the amount of time the moving object has spent on the path. Similarly, the studies in [11] and [12] detect hot spots without taking into account the road network restrictions. In [11], the authors address the problem of hot spot analysis over big trajectory data in a parallel and scalable way. Their approach is based on spatio-temporal partitioning of the 3D space in cells. The Getis-Ord index is employed and appropriately tailored to reveal hot spots in terms of spatio-temporal cells. Two algorithms are proposed: the THS algorithm defines the z-score of the cell under study depending to the cell values of the whole grid, which is computationally expensive. The aTHS algorithm as the authors mentions trade-off accuracy for efficiency in a controlled manner and ignore the contribution of cells located out side a predefined area of influence. Finally, Qiao et al. [12] introduce a cloud-based analytical framework for big mobile data from the view of user mobility in densely populated areas, collected from 2G/3G/4G networks.

Our work differs from these approaches as we consider road segments as candidate hot spots and adapt a well-known hot spot index to be applicable on temporal graphs that represent the road network for different temporal intervals. In this way, we capture the effect of spatial and temporal proximity of neighboring congested road segments. This is a differentiating factor compared to related works that operate over trajectories, as they produce hot spots at different level of granularity (e.g., routes, paths, grid cells) and using other metrics. To the best of our knowledge ours is the first work that shows how to apply hot spot analysis via Getis-Ord in road networks.

3. Problem Formulation

3.1. Preliminaries

We consider a directed graph $G(V, E)$ that represents the road network, where an edge $e_i \in E$ corresponds to a road segment and is defined by two vertices $v, v' \in V$, which represent crossroads. The edge e_i is directed, indicating the flow of movement on the road network from v to v'. We use the terms edge and road segment interchangeably.

To model the effect of time on traffic congestion, we consider a sequence $G_T = \{G_1, G_2, \ldots, G_m\}$ of *temporal snapshots* of the graph G, where each snapshot corre-

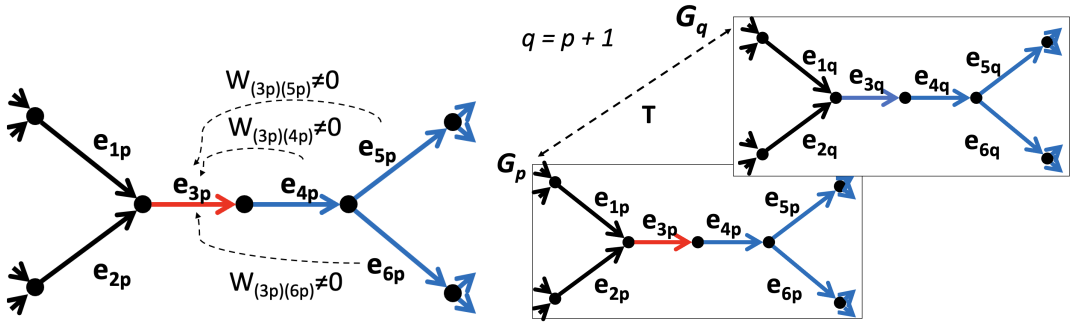

Figure 1: Left: Example of an edge e_{3p} of temporal snapshot G_p, which is affected only by subsequent edges e_{4p}, e_{5p} and e_{6p} (in blue). Right: Two temporal snapshots of G_p and G_q ($q = p+1$), where e_{3p} in G_p is affected also by the edges $e_{3q}, \ldots, e_{6q}$ of the next snapshot G_q.

sponds to a pre-defined temporal duration T. A snapshot G_p corresponds to an interval $[t_p, t_{p+1})$ of duration T, i.e., $t_{p+1} - t_p = T$. For example, if T is set to 5 minutes, one such interval could be 9:00 am – 9:05 am. A temporal snapshot G_p of the graph G is used to represent the traffic for the specific time interval $[t_p, t_{p+1})$. Fig. 1 illustrates the concept of temporal snapshots of a graph that represents the road network.

3.2. Problem Formulation

The problem of *hot spot analysis* addressed in this work is to identify statistically significant road segments that indicate hot spots, due to traffic congestion. An edge e_i at snapshot G_p, denoted $e_{ip} \in E(G_p)$, where $E(G_p)$ represents the edges of the snapshot G_p, is associated with an *attribute value* x_{ip} that indicates traffic congestion on e_i during the temporal interval of G_p. As such, whether an edge e_{ip} is considered hot spot is a function of the edge's attribute value x_{ip}, but also of other neighboring edges' attribute values. By neighboring edges, we mean both spatially (nearby edges on the graph G) and temporally (edges at previous/next temporal snapshots).

To express this dependence on neighboring edges, we adjust a commonly used function, namely the Getis-Ord statistic [13]. To the best of our knowledge, Getis-Ord has been used so far to discover hot spots in cell-based partitions for spatial (2D) and spatio-temporal (3D) domain, for point [14, 5] and trajectory data [11]. Our work extends the applicability of Getis-Ord for hot spot discovery in graphs representing the road network. Thus, for any edge e_{ip} of a snapshot G_p of G we adapt Eq. 1 and define the Getis-Ord value of the i-th edge of the p-th temporal snapshot:

$$G^*_{ip} = \frac{\sum_{j=1}^{|E|}\sum_{q=1}^{m} w_{(ip),(jq)}x_{jq} - \overline{X}\sum_{j=1}^{|E|}\sum_{q=1}^{m} w_{(ip),(jq)}}{S\sqrt{\frac{[n\sum_{j=1}^{|E|}\sum_{q=1}^{m} w^2_{(ip),(jq)} - \sum_{j=1}^{|E|}\sum_{q=1}^{m} w^2_{(ip),(jq)}]}{n-1}}} \quad (2)$$

where $n = |E| \cdot m$ is equal to the total number of edges, x_{jq} is the attribute value for edge e_j at temporal snapshot G_q, $w_{(ip),(jq)}$ is the weight imposed by edge e_{jq} to edge e_{ip}, and:

$$\overline{X} = \frac{\sum_{j=1}^{|E|}\sum_{q=1}^{m} x_{jq}}{n} \quad (3)$$

$$S = \sqrt{\frac{\sum_{j=1}^{|E|}\sum_{q=1}^{m} x^2_{jq}}{n} - (\overline{X})^2} \quad (4)$$

The weight $w_{(ip),(jq)}$ between two edges e_{ip} (edge e_i at snapshot G_p) and e_{jq} (edge e_j at snapshot G_q) indicates the influence of edge e_{jq} to edge e_{ip}. Intuitively, if a road segment is congested, this also affects other road segments in its vicinity. More precisely, in this paper, we assume that if a road segment e is congested, this affects (or will affect in the near future) the *previous* road segments, i.e., those leading to e. For example, in Fig. 1 (left), the edge e_{3p} (in red color) is affected by the weights of edges e_{4p}, e_{5p} and e_{6p} (in blue color), whereas all other weights are considered equal to zero $w_{(3p)(1p)} = w_{(3p)(2p)} = 0$.

Moreover, when considering different temporal snapshots of the graph G, the congestion of edges in snapshot G_{i+1} affect edges of previous and next temporal snapshots $\{\ldots, G_{i+2}, G_i, G_{i-1}, \ldots\}$. In Fig. 1 (right), edge e_{3p} at snapshot G_p is affected by edges $e_{3q}, \ldots, e_{6q}$ at G_q.

According to the above, the problem of hot spot analysis in road networks is to identify the k most statistically significant road segments (edges) according to the Getis-Ord statistic and can be formally stated as follows.

Problem 1. *(Hot spot analysis in road networks) Given a sequence G_T of temporal snapshots of a directed graph $G(V, E)$ representing the road network, find the top-k edges E_k of G_T based on the Getis-Ord statistic G^*_{ip}, such that: $G^*_{ip} \geq G^*_{jq}$, $\forall e_{ip} \in E_k, e_{jq} \in (\bigcup_{r=1}^{m} E(G_r)) - E_k$.*

Intuitively, for each road segment we compute a value that indicates if it is a hot spot. This value depends on the traffic congestion of the road segment itself, but also on the traffic congestion of subsequent road segments (as congestion is propagated backwards). Moreover, it depends on the traffic congestion of these road segments in the next temporal snapshots of the network, as illustrated in Fig. 1.

The novelty of our work lies in the identification of hot spots in road networks using a statistically significant metric. In contrast, existing works (Section 2) either use ad-hoc ways to compute hot spots or compute statistically significant hot spots as 2D/3D cells of predefined size. Adapting the Getis-Ord statistic to become applicable for graphs is the main technical contribution of our work. The challenge relates to the definition of neighboring edges over multiple temporal snapshots of the same graph.

4. Hot Spot Analysis

This section presents our approach for hot spot analysis in road networks. The input is a data set of GPS records,

each corresponding to a specific vehicle (based on identifier *vehID*), a timestamp t, as well as the longitude x and latitude y values. Also, a graph $G(V, E)$ is available that represents the road network, which consists of road segments, their geometry, as well as additional information, such as direction, speed limit, etc. Such a graph can be typically obtained from external sources, such as OpenStreetMap.

4.1. Map-matching

Data collected from GPS sensors typically contain small errors, thereby making the mapping of GPS coordinates to the underlying road network hard. This problem is typically solved by applying a map-matching algorithm, which is responsible for mapping each pair of GPS coordinates (longitude, latitude) to another point located on the road network. Several sophisticated algorithms for map-matching have been developed, such as [15, 16].

As the problem of map-matching is orthogonal to the one we are trying to solve, we adopt a simplistic approach where each pair of GPS coordinates is assigned to the nearest edge. In this way, each GPS record is enriched with an edge (road segment) of the graph, the coordinates of the two points that define the road segment, its length, as well as with other available information such as the speed limit.

4.2. Estimating an Edge's Traffic Congestion

In order to compute the Getis-Ord value of an edge e_{ip}, the first step is to compute the edge's attribute value x_{ip}, an indicator of traffic congestion. This is computed as the average value of contributions from each vehicle that reported its position on e_{ip} during the timespan of G_p. In the same spirit as in [17], we define this contribution as:

$$1 - \frac{v_{obs}}{v_{ffs}}$$

where v_{obs} is the observed speed of a vehicle on edge e_{ip}, whereas v_{ffs} is the free flow speed for edge e_i. One way to compute v_{obs} is to exploit the enter (t_{in}) and exit (t_{out}) time of the vehicle on edge e_{ip}. Thus, we have:

$$v_{obs} = \frac{l}{t_{out} - t_{in}}$$

where l is the length of the edge, and t_{in} (t_{out}) is the entry (exit) time. Unfortunately, for a vehicle that provides its position on edge e_{ip} (after map-matching), we do not know the exact entry and exit time. Therefore, we estimate these values, in particular given two successive positions we compute the entry time t as follows:

$$t = t_1 + d_1 \cdot \frac{t_2 - t_1}{d_1 + d_2}$$

where t_1 and t_2 correspond to the two timestamps, while d_1 and d_2 are the distances of the two points from the entry node, as shown in Fig. 2. In this way, we can estimate the entry/exit times for each edge.

4.3. Defining the Weights

Given two edges e_{ip} and e_{jq}, where e_i precedes e_j on graph G, the weight $w_{(ip),(jq)}$ indicates the influence of traffic congestion of edge e_{jq} to edge e_{ip}. The weight is a function of the distance between the edges, as an edge e_{jq} that is in close proximity with e_{ip}, has a stronger effect on e_{ip}'s

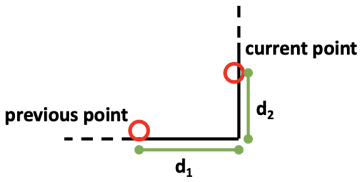

Figure 2: Example showing how to compute the entry time for an edge given two positions on successive edges.

congestion in comparison with another edge e'_{jq} that is further away. As in [18], we use a Gaussian kernel as weight function, defined as follows:

$$w_{(ip),(jq)} = exp(-\frac{d^2_{(ip),(jq)}}{h^2})$$

where h is the kernel's bandwidth and $d_{(ip),(jq)}$ denotes the distance between edges e_{ip} and e_{jq}. The distance function takes into account both spatial as well as temporal proximity. As such, we define the distance as a linear combination of spatial and temporal distance:

$$d^2_{(ip),(jq)} = \mu^S (d^S_{(ip),(jq)})^2 + \mu^T (d^T_{(ip),(jq)})^2$$

where μ^S, μ^T are scale factors for spatial and temporal distance respectively, whereas $d^S_{(ip),(jq)}$ is the spatial distance and $d^T_{(ip),(jq)}$ the temporal distance. The spatial distance $d^S_{(ip),(jq)}$ is measured as the number of edges between e_i and e_j on graph G. The temporal distance is the difference between temporal snapshots, e.g., edges at snapshots G_p and G_{p+1} have temporal distance equal to 1.

5. Parallel Algorithms

In this section, we present two data-parallel algorithms designed and implemented in Apache Spark that compute hot spots over very-large collections of GPS records in a scalable way. The first algorithm is an exact algorithm, whereas the second computes approximate results but with high accuracy and much faster.

In Apache Spark, the main data structure is the DataFrame, and its predecessor the RDD (Resilient Distributed Dataset) [19]. RDDs are immutable, distributed collections of records, which support transformations and actions. Transformations produce new RDDs from a given RDD, with notable examples: map, flatMap, mapValues, filter, etc. Actions perform some computation on an RDD and produce a new result (e.g., count). Our algorithms are implemented in Spark to support parallel processing.

5.1. Pre-processing

In the pre-processing phase, we perform map-matching (see Section 4.1) and we split the data of each vehicle in a set of trajectories. For each vehicle, we create a new trajectory when we observe a temporal gap of δt in the timestamps of reported positions. The value of δt is a parameter that depends on the sampling rate of the data set at hand.

As a result, the output of the pre-processing phase is a set of records in the following format: (vehID, trajID, x, y, t, e_i, length, speed, v, v'), where each GPS record (vehID, x, y, t) of the original data set is enriched with information

Algorithm 1 Computation of traffic congestion on edges

1: **Input:** The result of pre-processing: `inputfile`, number of Apache Spark partitions P, temporal parameters: t_{min}, t_{max}, T, m
2: **Output:** RDD[(e_i,t_p),x_{ip}]
3: **function**
4: dataRDD = sc.load(`inputfile`)
5: attrRDD = dataRDD.**groupByKey**().
6: **mapValues**(sort_grouped_values).
7: **flatMap**(lambda group: calc_attribute_value(group, t_{min}, t_{max}, T, m)).
8: **partitionBy**(P, lambda k: hash_partitioner).
9: **mapValues**(lambda x: (x,1)).
10: **reduceByKey**(lambda x, y: (x[0] + y[0], x[1] + y[1])).
11: **mapValues**(lambda x: x[0] / x[1])
12: **end function**

about the road segment (e_i, length, speed, v, v') and each record is assigned with a trajectory identifier (trajID). The values v and v' correspond to the start and end vertices of edge e_i, while speed is the speed limit which is typically provided in the road network data set and we use it as free flow speed v_{ffs}.

5.2. The Exact Algorithm

The main stages of our algorithm are summarized below:

- **Stage 1:** A set of transformations and aggregations is applied to the input data set to assign a traffic congestion value x_{ip} to each edge e_{ip}.
- **Stage 2:** The road network is represented in the form of a graph and the mean value ($\overline{X}$) and standard deviation (S) of the traffic congestion variable are computed. This data is assigned to a broadcast variable in order to become available to all the nodes of the cluster.
- **Stage 3:** The G^*_{ip} z-score for each edge e_{ip} is computed.
- **Stage 4:** The results of the previous step are filtered for a level of confidence and the top-k edges with the highest G^*_{ip} z-score are selected.

It should be noted that stages 1 and 3 are the time-consuming steps, therefore we delve into their details next. Stage 3 is the dominant cost, while stage 1 practically involves reading data from disk, but it is still much lower than the cost of stage 3.

Algorithm 1 describes how the traffic congestion values are computed (i.e., stage 1). First, we create an RDD of key-value pairs (known as PairRDD) with trajID as key (groupByKey, line 5) and value the records that correspond to this trajectory in sorted temporal order (mapValues, line 5). Then, for subsequent edges of each trajectory we compute the congestion value per vehicle per edge (using function calc_attribute_value()) and we create a new PairRDD with a composite key that consists of the edge and the temporal snapshot, while the value is the congestion value (flatMap, line 7). Subsequently, we partition the records based on edge using hash partitioning (line 8). In other words, all records that have been mapped to the i-th edge e_{ip} at snapshot G_p are assigned to the same partition. This allows the computation of the average congestion value x_{ip} per edge based on the contributions of all vehicles (lines 9–11). It should be noted that the function *calc_attribute_value()* performs the computation of x_{ip}, as described in Section 4.2.

Algorithm 2 Computation of Getis-Ord index

1: **Input:** The attrRDD: RDD[(e_i,t_p),x_{ip}], number of Apache Spark partitions P, temporal parameters: t_{min}, t_{max}, T, m, bandwidth h, output of stage 2: $\overline{X}$, S, G
2: **Output:** RDD[(e_i,t_p),G^*_{ip}]
3: **function**
4: GetisOrdRDD = attrRDD.**flatMap**(lambda x : GetisOrdCalculations(x, t_{min}, t_{max}, T, m, h, $\overline{X}$, S, G))).
5: **partitionBy**(P, lambda k: hash_partitioner).
6: **reduceByKey**(lambda x, y: (x[0] + y[0], x[1] + y[1], x[2] + y[2])).
7: **mapValues**(lambda x: Gi_formula(x, $\overline{X}$, S))
8: **end function**

Then, in stage 2, we compute the aggregate values necessary for the computation of Getid-Ord. These include the mean $\overline{X}$, the standard deviation S, and the total number of edges n for all temporal snapshots of the graph G. This information together with the graph G is broadcast to all worker nodes in the cluster, so as to be available during the computation of Getis-Ord.

In stage 3, we compute the Getis-Ord index G^*_{ip} per edge e_{ip} and temporal snapshot G_p. Algorithm 2 provides the pseudocode for the computation of Getis-Ord values. First, using flatMap (line 4), we transform attrRDD into a new RDD with key the edge e_{ip} and value a triple:

$$w_{(ip),(jq)}, w^2_{(ip),(jq)}, w_{(ip),(jq)}x_{jq}$$

that corresponds to another edge e_{jq}. This computation is performed by function *GetisOrdCalculations()*, which takes into account neighboring edges e_{jq} to the current edge e_{ip} and computes their contribution to e_{ip} in terms of weight, squared weight and weight times attribute value, which are the basic constituent parts for computing Getis-Ord. Notice that this function takes as parameters the information about temporal snapshots, the kernel's bandwidth h, as well the broadcast data. Then, we perform hash partitioning based on edge in order to accumulate all records that correspond the same edge for the same timestamp (line 5), in order to compute the aggregate values (line 6):

$$\sum_{j=1}^{|E|}\sum_{q=1}^{m} w_{(ip),(jq)}$$

$$\sum_{j=1}^{|E|}\sum_{q=1}^{m} w^2_{(ip),(jq)}$$

$$\sum_{j=1}^{|E|}\sum_{q=1}^{m} w_{(ip),(jq)}x_{jq}$$

Thereafter, we compute the Getis-Ord value G^*_{ip} per edge e_i and per temporal snapshot G_p using function *Gi_formula()* (line 7).

Finally, in stage 4, the edges are ordered based on the computed Getis-Ord value and only the top-k edges are returned. Obviously, k is a parameter that is set according to the application requirements.

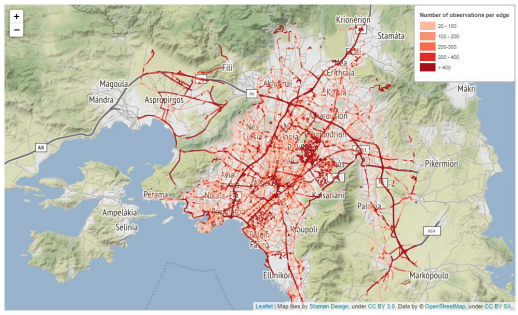

Figure 3: The data set used in the experimental study.

5.3. Discussion

From the analysis of the exact algorithm we can derive useful information about its complexity with respect to the input parameters. We focus on the number of records that need to be processed, as they influence the processing cost.

In Algorithm 1, the size of dataRDD is $O(N)$, where N denotes the number of GPS records. Then, the algorithm builds the attrRDD which contains one record per edge per temporal snapshot. If $|E|$ is used to denote the number of edges of graph G (the road network) and m denotes the number of temporal snapshots of G, then attrRDD contains $O(|E| \cdot m)$ records. As such, its size is linearly dependent on the size of the road network (in edges) and the number of temporal snapshots.

In Algorithm 2, the attrRDD is given as input and another RDD called GetisOrdRDD is built. For a given temporal snapshot, this contains $O(|E|^2)$ records because it combines each edge with its neighboring edges with non-zero congestion value (in worst case with all edges). When other temporal snapshots are considered too, the number of records increases exponentially. Distributing these records and aggregating per edge is the main factor that affects the processing cost of the exact algorithm.

5.4. The Approximate Algorithm

The previous algorithm computes the exact Getis-Ord z-scores, but it is computationally expensive because every edge e_{jq} with non-zero congestion value (x_{jq}) has an influence on the z-score of all other edges. However, edges at high distance (in spatial or temporal terms) are expected to have extremely small impact on the computed z-score.

Motivated by this observation, we propose the use of an approximate algorithm that takes as input a cut-off threshold c, which determines the subset of neighboring edges that will be considered when computing the Getis-Ord values of a given edge. In spatial terms, c corresponds to number of hops between two edges. In temporal terms, c corresponds to the number of (previous/next) temporal snapshots that should be considered. In this way, we limit the number of records that need to be processed and do not allow this number to increase exponentially. In turn, this drastically reduced number of records that need to be processed makes the algorithm work much faster, at the expense of slightly inaccurate results.

As shown in our experimental study, the approximate algorithm offers an interesting trade-off between performance and accuracy. Thus, we can compute hot spots with high accuracy but in a much shorter execution time.

Parameters	Values
Size of the data set (x10^6 GPS records)	2.5, 5, **7.4**
Duration (T) of temporal snapshots (in hrs)	**24**, 48, 168
Number of partitions (P)	6, 8, 18, 60, **120**
Cut-off threshold (c) in hops	6, 12, 25, ∞

Table 1
Experimental parameters and values (default values in bold).

6. Experimental Evaluation

We evaluate the performance of our approach for urban hot spot analysis. We implemented both algorithms in PySpark, using Apache Spark 3.1.1 Core API.

6.1. Experimental Setup

Platform. We deployed our code in Google Cloud Platform (GCP). The platform supports the process of big data in Apache Spark. The initiated cluster runs on Debian 10, while we used the version Spark 3.1.1 in Hadoop 3.2.2. The cluster consisted of 4 nodes in total, 1 Master and 3 Workers. Each node has 2vCPU 13GB RAM, 500GB hard drive and 375GB SSD.

Data Set. We employed a real data set of anonymized GPS records collected by a fleet management provider. The data was collected over a a period of five months from July to November 2018, consisting of 368,977 individual vehicle trajectories moving at the Metropolitan Area of Athens in Greece (Fig. 3). This data set is 1.1GB in total size and contains approximately 7.4M (million) GPS records. We also create two subsets of size 5M and 2.5M records. After pre-processing (Section 5.1), each record consists of the fields: (vehID, trajID, x, y, t, e_i, length, speed, v, v'). Even though the specific data set is sparse and medium-sized, it is still a real-world data set that allows the evaluation of our approach.

Metrics. Our main evaluation metric was the execution time needed for each individual stage of the algorithm to be completed. The execution time is measured in seconds.

Evaluation Methodology. The efficiency of our algorithm was assessed based on four parameters: (a) the size of the data set, (b) the temporal window T, which determines the number of temporal snapshots, (c) the number of Apache Spark partitions P, and (d) the cut-off threshold c, used only by the approximate algorithm (for the exact $c = \infty$), that defines the neighboring edges in the axes of time and space that contribute in Getis-Ord z-score, measured in hops. After different tests, we also set the kernel's bandwidth $h = 8$ for the exact algorithm, while $h = \frac{2}{3} \cdot c$ for the approximate algorithm. Our experimental setup is summarized in Table 1.

6.2. Results for the Exact Algorithm

Varying the size of the data set. In Fig. 4 (left), we demonstrate the performance of the exact algorithm when increasing the data set size. We measure the time of stage 3, which is the most time-consuming part of the algorithm. As the number of observations to be processed increases from 2.5M to 7.4M, so does the total execution time of the algorithm but not linearly. This is due to the fact that the larger data set implies more temporal snapshots, hence more edges in

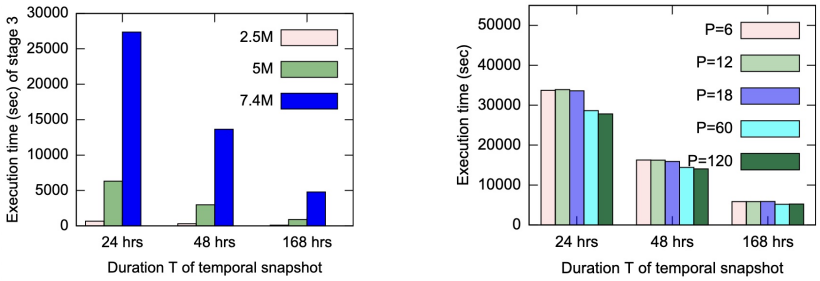

Figure 4: Left: execution time of stage 3 of the exact algorithm when increasing the data set size and varying the temporal duration T of snapshots. Right: total execution time for the data set of 7.4M records for different number of partitions P.

total, which all need to be processed for the computation of the Getis-Ord value of an edge.

Varying the duration T of temporal snapshots. The temporal duration T of 24, 48 and 168 hours for a period of five months leads to the generation of 153, 27 and 22 non-empty temporal snapshots respectively. Therefore, the fewer the temporal partitions, the smaller the overall execution time. In Fig. 4 (left) the temporal partitioning that comes from $T = 48$ hrs time window is twice as efficient in terms of execution time as the $T = 24$ hrs time window. The $T = 168$ hrs time window is about seven times more efficient than $T = 24$ hrs, indicating a linear relationship between execution time and temporal duration T for the data set of 2.5M and 5M records.

Varying the number P of Spark partitions. Fig. 4 (right) shows the total execution time of the exact algorithm and the time of stage 3 for different numbers of Spark partitions. The first observation is that the execution time of stage 3 practically determines the overall execution time. Regarding the effect of using a higher number of partitions, this seems to reduce execution time. Using more partitions results in smaller subsets of data to be processed, which require less processing time. However, the reduction is not that big (about 15% when using 120 partitions for $T = 24$hrs).

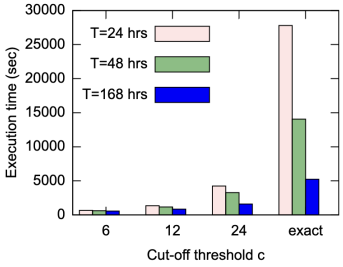

Figure 5: Execution time of the approximate algorithm for different cut-off values c and temporal duration T.

6.3. Results for the Approximate Algorithm

Fig. 5 demonstrates that the overall execution time is significantly reduced for smaller values of c, since the weight function is calculated only for the edges of the graph that are c hops away from an edge both spatially and temporally. Compared to the exact algorithm, the gain in execution time can be of orders of magnitude. For example, for $T = 24$ hrs, the exact algorithm needs 27, 799 sec, whereas the approximate algorithm with $c = 6$ takes only 659 sec. This shows that our approximate algorithm that uses the cut-off threshold can make the discovery of hot spots practically applicable for extremely large data sets.

However, the remaining question is how accurate are the results of the approximate algorithm? Fig. 6 shows the hot spots discovered by the approximate and the exact algorithm. This demonstrates visually that the results of the experiment for $c = 12$ are similar to those obtained when applying the exact algorithm.

To complement the visual evaluation, we also performed a quantitative study. We obtained a ranked list of the top-100 edges in terms of traffic congestion for $c = 12$ and for the exact algorithm. Then, we computed the Spearman's rank correlation for these two lists, in order to assess if the same ordering is observed (i.e., if the ranking for $c = 12$ resembles the ranking obtained by the exact algorithm). If the two rankings were identical, we would obtain a value of 1. In our case, we obtained a value of 0.82 for the Spearman's rank correlation. This indicates that the two lists are highly correlated, or (put differently) that the hot spots discovered for $c = 12$ are very similar to the ones of discovered by the exact algorithm. In the case of the results using $c = 6$, there is less sensitivity in the recognition of extreme values, while in the case of $c = 24$ there is hypersensitivity resulting in more edges being recognized as hot spots. Our experiments indicate that using a cut-off threshold around $c = 12$ provides a nice trade-off between performance and accuracy.

7. Conclusions

In this paper, we formulate the problem of hot-spot discovery in road networks. We proposed an exact and an approximate algorithm for hot spot discovery, which are implemented as data-parallel algorithms on top of Apache Spark. Our experiments using a real-life data set indicate that the approximate algorithm provides a viable solution that trades performance for accuracy. In the future, we will apply our algorithms to other real-life data sets of larger scale.

Acknowledgements

This research has received funding from the European Union's funded Project EMERALDS under grant agreement no 101093051.

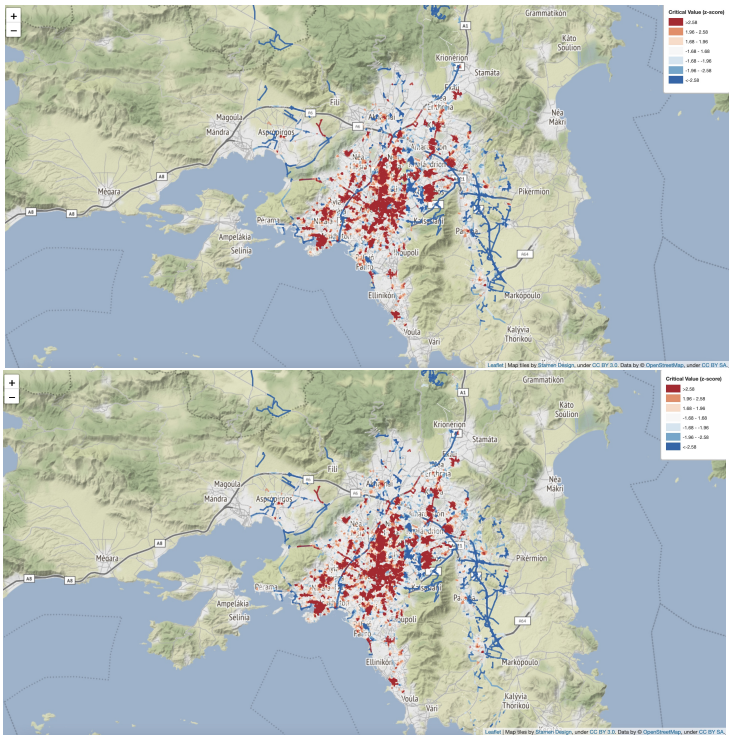

Figure 6: Hot spots discovered by the approximate algorithm for $c = 12$ (left), and for the exact algorithm, i.e., $c = \infty$ (right).

References

[1] E. Eftelioglu, X. Tang, S. Shekhar, Geographically robust hotspot detection: A summary of results, in: Proc. of ICDMW'15, 2015, pp. 1447–1456.

[2] E. Eftelioglu, S. Shekhar, D. Oliver, X. Zhou, M. R. Evans, Y. Xie, J. M. Kang, R. Laubscher, C. Farah, Ring-shaped hotspot detection: A summary of results, in: Proc. of ICDM'14, 2014, pp. 815–820.

[3] X. Tang, E. Eftelioglu, S. Shekhar, Elliptical hotspot detection: A summary of results, in: Proc. of BigSpatial'15, 2015, p. 15–24.

[4] Y. Xie, S. Shekhar, Significant DBSCAN towards statistically robust clustering, in: Proc. of SSTD'19, 2019, p. 31–40.

[5] P. Nikitopoulos, A.-I. Paraskevopoulos, C. Doulkeridis, N. Pelekis, Y. Theodoridis, BigCAB: Distributed hot spot analysis over big spatio-temporal data using Apache Spark, in: Proc. of SIGSPATIAL'16, 2016.

[6] X. Tang, E. Eftelioglu, D. Oliver, S. Shekhar, Significant linear hotspot discovery, IEEE Trans. Big Data 3 (2017) 140–153.

[7] X. Tang, E. Eftelioglu, S. Shekhar, Detecting isodistance hotspots on spatial networks: A summary of results, in: Proc. of SSTD'17, 2017, pp. 281–299.

[8] M. Häsner, C. Junghans, C. Sengstock, M. Gertz, Online hot spot prediction in road networks, in: Proc. of BTW'11, 2011, pp. 187–206.

[9] X. Li, J. Han, J. Lee, H. Gonzalez, Traffic density-based discovery of hot routes in road networks, in: Proc. of SSTD'07, 2007, pp. 441–459.

[10] D. Sacharidis, K. Patroumpas, M. Terrovitis, V. Kantere, M. Potamias, K. Mouratidis, T. Sellis, On-line discovery of hot motion paths, in: Proc. of EDBT'08, 2008.

[11] P. Nikitopoulos, A.-I. Paraskevopoulos, C. Doulkeridis, N. Pelekis, Y. Theodoridis, Hot spot analysis over big trajectory data, in: Proc. of Big Data'18, 2018, pp. 761–770.

[12] Y. Qiao, Y. Cheng, J. Yang, J. Liu, N. Kato, A mobility analytical framework for big mobile data in densely populated area, IEEE Trans. on Veh. Techn. 66 (2017) 1443–1455.

[13] J. K. Ord, A. Getis, Local spatial autocorrelation statistics: Distributional issues and an application, Geographical Analysis 27 (1995) 286–306.

[14] G. Makrai, Efficient method for large-scale spatio-temporal hotspot analysis, in: Proc. of SIGSPATIAL'16, 2016.

[15] C. Y. Goh, J. Dauwels, N. Mitrovic, M. T. Asif, A. Oran, P. Jaillet, Online map-matching based on hidden markov model for real-time traffic sensing applications, in: Proc. of ITSC'12, 2012, pp. 776–781.

[16] Y. Lou, C. Zhang, Y. Zheng, X. Xie, W. Wang, Y. Huang, Map-matching for low-sampling-rate GPS trajectories, in: Proc. of SIGSPATIAL'09, 2009, pp. 352–361.

[17] H. Xiong, A. Vahedian, X. Zhou, Y. Li, J. Luo, Predicting traffic congestion propagation patterns: A propagation graph approach, in: Proc. of IWCTS@SIGSPATIAL'18, 2018, pp. 60–69.

[18] B. Wu, R. Li, B. Huang, A geographically and temporally weighted autoregressive model with application to housing prices, Int. J. Geogr. Inf. Sci. 28 (2014) 1186–1204.

[19] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauly, M. J. Franklin, S. Shenker, I. Stoica, Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing, in: Proc. of NSDI, USENIX Association, 2012, pp. 15–28.