UNIVERSITY OF PIRAEUS

DEPARTMENT OF DIGITAL SYSTEMS

POSTGRADUATE PROGRAMME INFORMATION SYSTEMS & SERVICES

OFFLINE HOTSPOT ANALYSIS OVER ROAD NETWORK TRAJECTORIES

POSTGRADUATE THESIS

of

ALEXANDROS IOANNIS VOSSOS

Supervisor: Christos Doulkeridis
Professor Uni.Pi.

Athens, 2020

UNIVERSITY OF PIRAEUS
DEPARTMENT OF DIGITAL SYSTEMS
POSTGRADUATE PROGRAMME INFORMATION SYSTEMS & SERVICES

OFFLINE HOTSPOT ANALYSIS OVER ROAD NETWORK TRAJECTORIES

POSTGRADUATE THESIS

of

ALEXANDROS IOANNIS VOSSOS

Supervisor: Christos Doulkeridis
Professor Uni.Pi.

Approved by

Date:……

(Signature)	*(Signature)*	*(Signature)*
.....	.....	.....
Christos Doulkeridis Professor Uni.Pi.	Maria Halkidi Professor Uni.Pi.	Orestis Telelis Professor Uni.Pi.

Athens, 2020

(Signature)

.....

ALEXANROS IOANNIS VOSSOS

Postgraduate student of the Department of Digital Systems Uni.Pi.

Περίληψη

Βάση ενός συνόλου δεδομένων τροχιών οχημάτων σε ένα οδικό δίκτυο, οι αλγόριθμοι που παρουσιάζονται σε αυτή την εργασία στοχεύουν να ανακαλύψουν στατιστικά σημαντικά hotspots στο εν λόγω οδικό δίκτυο. Αυτή η προσέγγιση βρίσκει εφαρμογή στην βελτίωση οδικών δικτύων καθώς αυτή η ανάλυση μεγάλων συνόλων δεδομένων τροχιών οχημάτων που μπορούν να αποκαλύψουν στοιχεία για την απόδοση των εν λόγω δικτύων. Τα hotspots εντοπίζονται χρησιμοποιώντας μια προσαρμοσμένη έκδοση της μετρικής Getis-Ord G_i^*, το οποίο έχει παραδοσιακά χρησιμοποιηθεί για την ανάλυση σημειακών δεδομένων σε ένα δισδιάστατο πλέγμα. Η μετρική αυτή έχει προσαρμοστεί για την ανίχνευση hotspots σε μια τρισδιάστατη χωροχρονική απεικόνιση, σε μορφή γράφου, του οδικού δικτύου που υπάρχει στο σύνολο δεδομένων εισόδου. Αναπτύχθηκαν δύο αλγόριθμοι, ένας που παρέχει μια ακριβής λύση στο πρόβλημα και ένας που παρέχει μια προσέγγιση θυσιάζοντας ακρίβεια για κέρδος σε υπολογιστικό κόστος. Η αποδοτικότητα και η επεκτασιμότητα των αλγορίθμων ελέγχονται πειραματικά σε ένα σύνολο δεδομένων τροχιών στο οδικό δίκτυο της Ελλάδας.

Λέξεις Κλειδιά: Εύρεση Hotspot, στατιστική σημαντικότητα, χωροχρονικά δίκτυα, παράλληλη επεξεργασία, Apache Spark

Abstract

Given a dataset of vehicle trajectories on a road network the algorithms presented in this dissertation aim to discover statistically significant hotspots in said road network. This approach finds application in transportation engineering and the analysis of large collections of vehicle trajectories that can provide insights on the performance of road networks. Hotspots are identified using an adapted version of the Getis-Ord G_i^* statistic, which has traditionally been used for the analysis of point data over a 2D grid. The statistic has been adapted to detect hotspots on a 3D spatio-temporal graph representation of the road network present in the input dataset. Two algorithms were developed; one that provides an exact solution to the problem and one that provides an approximate solution trading off accuracy for computational cost. The efficiency and scalability of the algorithms are tested experimentally on a vehicle trajectory dataset in Greece's road network.

Keywords: Hotspot detection, statistical significance, spatial networks, parallel processing, Apache Spark

Table of Contents

1 Introduction 1

1.1 Big Mobility Data 1

1.2 Hotspot Analysis 1

1.3 Contribution 2

1.4 Dissertation Organisation 3

2 Related Work 4

2.1 Hotspots 4

2.2 Road Network Trajectories 5

3 Theoretical Background 7

3.1 Spark 7

3.2 Graph Representation of a Road Network 8

3.2.1 OSMnx 9

3.2.2 NetworkX 9

3.3 Getis-Ord G_i^* Statistic 9

4 Problem Formulation 11

4.1 Input Data 11

4.1.1 Map Matching 12

4.1.2 Trajectory ID 13

4.2 Theory and Implementation 14

4.2.1 Attribute Value 14

4.2.2 Congestion Hotspot Analysis 16

5 Distributed Offline Algorithms 19

5.1 Operations Overview 20

5.2 Calculating Attribute Values 20

5.2.1 Calculate_attribute Function 21

5.3 Calculating Broadcast Variables 22

5.4 Calculating z-scores Using the Getis-Ord Statistic 23

5.4.1 GetisOrdCalc Function 24

5.5 Approximate Algorithm 25

6 Experimental Evaluation 28

6.1 Evaluation Dataset 28

6.2 Outputs 29

6.3 Evaluation Methodology 29

6.4 Results 30

6.4.1 Attribute Values Captured 30

6.4.2 Varying the Size of the Dataset 30

6.4.3 Varying r 32

6.4.4 Varying the Temporal Step Size 33

6.4.5 Hotspot Visualisation 34

7 Conclusion 38

7.1 Future Work 38

8 References 40

1

Introduction

1.1 Big Mobility Data

The expansion of GPS enabled devices, from smartphones used by individuals for navigation to GPS trackers installed by companies on their vehicles, has led to the creation of huge amounts of mobility data. Mobility data is a subcategory of big spatio-temporal data, which are characterized by the spatial and temporal dimensions being particularly significant. Spatio-temporal data includes trajectory data, geotagged tweets/posts, check-ins on social media, police reports on crimes/accidents, disease outbreak reports, and many more. Analysing large volumes of spatio-temporal data can reveal insights that would have otherwise gone unnoticed. Examples include notifying drivers in real-time of road blockages due to accidents, tracing back the epicentre of a disease outbreak, finding dangerous parts of cities and many more. Due to the large volume of this data extracting such insights requires specialised frameworks and algorithms that can scale with the size of the data.

1.2 Hotspot Analysis

A useful operation that can be applied on spatio-temporal data is hotspot analysis, an analysis technique that identifies clustering on spatial data. A statistically significant hotspot can be defined for a feature when it has a high value and is surrounded by other features with high

values as well. Hotspot detection was originally used for the study of point distributions in space where the distribution of points is compared against the map average. Hotspot analysis can be used on most domains of mobility data. For example, in aviation a high concentration of air traffic in an area can lead to the implementation of specific regulations. In marine data hotspots can indicate areas that are fished excessively. In road networks speeds lower than anticipated can be used to detect congestion hotspots.

1.3 Contribution

This thesis deals with hotspot analysis on big trajectory data of vehicles on a road network. The road network Is modelled as a directed graph were intersections are nodes and the roads between them are directed edges. The temporal aspect of the data can be divided into temporal steps, thus creating a layered graph where each graph layer represents the graph for a specified length of time. Our goal is to identify edges that are congestion hotspots, namely that have a statistically significant high value of the congestion parameter. To this goal we employ the Getis-Ord statistic which calculates z-scores and p-values for each feature in the dataset. These values indicate clusters of either high or low values. Hotspot features are associated with high z-scores and small p-values. While a feature with a high value might be worth investigating it is not necessarily a statistically significant hotspot. For that it must have a high value and the neighbouring features will also have high values. A statistically significant z-score is produced when the sum of the feature and its local neighbours is different enough from the sum of all features that it is not a result of random chance. The Getis-Ord statistic is generally used for 2D spatial data and deals with point data rather than trajectories. We are, however, dealing with road network trajectories, i.e., temporally sorted collections of spatio-temporal data that follow strict paths and the Getis-Ord statistic will need to be adjusted for that.

The Getis-Ord statistic is therefore modified to work with road network trajectory data. The goal of this modification is to capture the amount of congestion experienced by each edge in its attribute value. The modified statistic is then used in two parallel and scalable algorithms, an exact and an approximate one, for computing hotspots in the edges of the spatio-temporal graph of the road network. The exact algorithm calculates the z-scores for the edges in parallel and returns an exact result set. Spatially, the algorithm accounts for the directionality of the graph as only out-neighbours are considered. This is due to our intuition that an edge is likely to experience congestion if the edges after it are also congested. Temporally, both previous and following temporal partitions are considered. The exact algorithm considers, for each edge, all out-neighbours for all temporal partitions. The approximate algorithm calculates the z-scores for the edges in parallel and returns an approximate result set. Spatially, the algorithm only considers out-neighbours at a distance r (in number of edges) from the examined edge.

Temporally, only neighbouring partitions within the same distance r (in number of partitions) from the examined edge are considered.

In summary, this thesis makes the following contributions:

- The Getis-Ord statistic is adapted to the problem of trajectory hot-spot analysis on road networks. This allows it to function with temporally sorted collections of spatio-temporal data that follow strict paths instead of plain points.
- A parallel algorithm that calculates an exact solution to the problem is developed. It returns spatio-temporal edges with high z-scores that can be considered hotspots. This algorithm is further developed into an approximate solution that can trade off accuracy for computational cost.
- The algorithms have been developed with Apache Spark and their efficiency and scalability are evaluated on a dataset of vehicle trajectories in Greece.

1.4 Dissertation Organisation

The rest of the dissertation is organised as follows:

Chapter 2 presents a literature review on work related to the topics examined in this dissertation. In chapter 3 an overview of the theoretical background related to this work is presented. Chapter 4 deals with the formulation of the problem and the theoretical structure of its proposed solution. Chapter 5 deals with the implementation of the theoretical solution into an exact and an approximate algorithm. Chapter 6 presents the setup, methodology and results of the experimental evaluation of the algorithms. Chapter 7 sums up the dissertation and discusses future work.

2

Related Work

The related work can be separated into two main categories. Papers that relate to discovering hotspots and papers that deal with road network trajectories.

2.1 Hotspots

In [1] Nikitopoulos et al aim to discover hotspots over marine trajectory data. To this goal they employ a modified version of the Getis-Ord statistic where the contribution of a moving object to a cell's density is proportional to the time spent by the moving object in the cell. This is to allow the Getis-Ord statistic to work with trajectory data instead of point data. This thesis is based on their work and is meant to expand hotspot analysis over big trajectory data by considering trajectories on strictly defined road networks rather than arbitrarily divisible marine areas.

In [8] Häsner et al address the problem the online prediction of hotspots in road networks. Their approach (OPS) determines individual hotspot nodes by using an efficient heuristic to predict the traffic intensity at all nodes in the road network. Based on the hotspots nodes they then predict inferred subgraphs that represent hotspot regions. The main differences with our work is that they find hotspot nodes rather than edges and that the connected substructures they derive are then used to predict future hotspots.

In [3] Xie et al aim to detect statistically significant clusters of points. The use of DBSCAN allows them to find clusters of arbitrary shapes. They then utilise a Monte Carlo method to compute statistical significance of the model and a Dual-Convergence algorithm to accelerate the process. The main differences with our work are that they deal with point data rather than trajectories and are looking for clusters/ hotspots of arbitrary shapes rather than working on a strictly defined road network.

In [4] Tang et al given a spatial network and a collection of activities aim to find all shortest paths in the spatial network where the concentration of activities is statistically significantly high. This paper proposes novel models and algorithms for discovering statistically significant linear hotspots using the Significant Route Miner with both neighbour node filter, shortest path tree pruning, and Monte Carlo simulation speedup (SRM_TBD). The algorithms null hypothesis is that the density inside the path and outside are equal while the alternative hypothesis is that the density inside the path is greater than outside and hotspots are evaluated based on the density ratio of each edge. The algorithm finds all shortest paths that meet the minimum threshold of density ratio, the minimum level of statistical significance and the minimum distance threshold. They also implement dynamic segmentation, an algorithm that splits edges into sub edges at each activity and the weights for the new edges are recalculated. This enables them to detect hotspots within smaller parts of edges that might have otherwise been lost. The main differences with our work are that they deal with point data rather than trajectories and that they consider shortest path based hotspots rather than edge based.

2.2 Road Network Trajectories

In [5] Zygouras et al aim to discover commonly accessed corridors from GPS trajectories. Their approach starts with discovering frequent sets of locations observed often together. This is done by applying LDA to trajectory data to detect areas that share common traffic. Trajectories are then segmented into subtrajectories for each frequent set. The cells of the trajectory are searched against frequent sets for common cells. A subtrajectory is created using the cells between the first and last cells of the trajectory in common with the frequent set. If there are more than θ_{gap} consecutive cells unmatched the trajectory is split into two subtrajectories. Subtrajectories are grouped together using the hierarchical clustering algorithm on each frequent set. Clusters stop merging when the intra-cluster distances exceed a threshold. Dynamic time warping was used to distinguish direction of movement in the trajectories. Corridors are created for each frequent set given a cluster of similar trajectories. Low GPS sampling rate causes uncertainty in the individual trajectories so a directed edge-weighted graph is constructed for the cluster. This comprises of a set of vertices that correspond to cells and directed edges that connect adjacent grid cells. The weight is the likelihood of moving to that vertex and is calculated based on the visiting frequency of the cell and the most common direction of the moving objects. The most likely path is extracted using shortest path query and that is used to complete missing parts of trajectories. Each complete path is added to the list of corridors if the minimum distance between it and the already inserted corridors does not exceed a threshold. While both this paper and our work deal with GPS trajectories on road networks the goals are different as we are interested in hotspots rather than common corridors.

In [6] Krogh et al aim to compute novel and important information from GPS trajectories. Trajectories are used to calculate indicators such as: time to pass intersection, number of stops to pass a road, free flow speed, etc. The number of stops done by a vehicle can be an indicator of faulty loop detectors, poorly timed traffic lights or other problems of the road network. The number of stops together with the distance covered between them can indicate the maximum number of cars that can pass each cycle of an intersection. The free flow speed can be found by averaging the speed of the top x % fastest trajectories. These indicators can be used in conjunction with our work to modify the definition of what is considered a hotspot and how attribute values are calculated.

3

Theoretical Background

In this chapter an overview of the topics related to this work is presented. The main tools and models used in this work are Spark, Getis-Ord and Networkx.

3.1 Spark

For the development of our solution we have utilised Apache Spark. Spark is an open-source distributed general-purpose cluster-computing framework and it provides an interface for programming entire clusters with data parallelism and fault tolerance. It is generally used for solving problems that require analysing a huge volume of data. There are two basic tools of Spark. The Resilient Distributed Dataset (RDD) which is a read only collection of data items over a cluster of machines and it incorporates fault-tolerance. Spark Core is the foundation of the entire framework which enables transforming RDDs.

A Resilient Distributed Dataset (RDD) is the basic abstraction in Spark. It represents an immutable, partitioned collection of elements that can be operated on in parallel. RDDs can contain only values, key-value pairs or sequences. RDDs are fault tolerant by maintaining a record of the history of each RDD and how it was created which allows it to be recreated in the event of data loss. RDDs can contain any type of Python, Java, or Scala objects.

Internally, each RDD is characterized by five main properties:

- A list of partitions
- A function for computing each split
- A list of dependencies on other RDDs
- Optionally, a Partitioner for key-value RDDs (e.g. to say that the RDD is hash-partitioned)
- Optionally, a list of preferred locations to compute each split on (e.g. block locations for an HDFS file)

Spark Core provides distributed task dispatching, scheduling, and basic I/O functionalities. This interface invokes parallel operations available on all RDDs, such as map, filter, and persist. There are also operations that can only be applied on RDDs of key-value pairs, such as groupByKey and join. All operations are automatically available on any RDD of the right type. Parallel operations on RDDs are scheduled through Spark and executed in parallel on the cluster. The operations have RDDs as inputs and produce new RDDs as each RDD is immutable. Operations are lazy, meaning that execution does not occur until an action is called. In addition to the RDD structure Spark has two types of variables that are shared between cluster nodes. Broadcast variables which make read-only data available to all nodes and accumulators which are variables, such as sums or counters, that all nodes can add to.

Regarding the experiments performed in the evaluation section of this thesis, using the Spark platform was not necessary as the volume of data analysed is relatively small and could have been easily solved by conventional programming methods. The advantage of implementing our solution with Spark is that it allows us to easily scale the solution to larger volumes of data and the execution time could remain relatively constant by adding additional computers of similar specifications to the cluster.

3.2 Graph Representation of a Road Network

A graph is a set of objects and the relations between them. Objects are known as nodes or vertices and the relations between them are called edges. The edges can be directed or undirected and the graph is called an undirected or a directed graph accordingly.

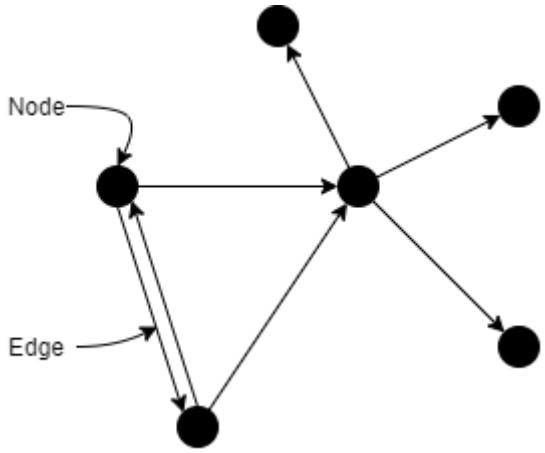


Fig. 1 A simple directed graph.

In this thesis we look at the road network as a directed graph where intersections are considered vertices and the roads between them are seen as edges. In this case the directionality of the graph is derived from the direction in which traffic is allowed to move on each street. In order

to work with the road network in the form of a graph the python libraries OSMnx and NetworkX were used.

3.2.1 OSMnx

OSMnx is a python library that allows downloading spatial geometries and modelling, projecting, visualising and analysing street networks from OpenStreetMap's APIs. OpenStreetMap is a free editable map of the world built by a community of mappers that contribute and maintain data about roads, trails, cafés, railway stations and other points of interest all over the world. Through the use of aerial imagery, GPS devices, and low-tech field maps contributors verify that OSM is accurate and up to date. Through OSMnx the relevant part of the map is downloaded as a network this includes useful road network information such as road speed and road type.

3.2.2 NetworkX

NetworkX is a Python library for working with graphs and networks. It supports data structures such as graphs, directed graphs and multigraphs. Nodes can be anything defined by the user and edges can contain data. It contains many standard operations that can be applied on graphs and networks.

3.3 Getis-Ord G_i^* Statistic

The Getis-Ord G_i^* statistic is a tool used for hotspot analysis which calculates z-scores for features [2]. It is defined as:

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

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

$$\bar{X} = \frac{\sum_{j=1}^{n} x_i}{n}$$

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

The resultant z-score indicates whether high or low valued features cluster spatially. A feature with a high value is not necessarily a statistically significant hotspot, it must have a high value and the neighbouring features will also have high values. A statistically significant z-score is

produced when the sum of the feature and its local neighbours is different enough from the sum of all features that it is not a result of random chance. When examining statistically significant positive z-scores, the larger the value, the more intense the clustering of high values (hotspot). For statistically significant negative z-scores, the smaller the value, the more intense the clustering of low values (cold spot).

4

Problem Formulation

In this chapter the individual parts of the problem are defined. The first subsection relates to the input data required by our solution, its format and all related assumptions. The second subsection analyses the development of the solution on a theoretical level.

4.1 Input Data

The core of this problem is a collection C of moving object trajectories t ∈C. Each trajectory represents the path followed by a motor vehicle on a road network. Trajectories are a sequence of temporally sorted data points p which contain 2D spatial coordinates (latitude and longitude) and a timestamp showing the position of the vehicle at a given time. There are two basic assumptions made regarding the trajectories. The first is that the trajectories have been created by motor vehicles that follow the flow of traffic and can reach the speed limit of the edge, e.g. cars, taxis, trucks and not by bicycles or pedestrians. The second assumption is that the sampling rate of the trajectory is high, i.e., on average creating data points every few (1-10) seconds. Additional information regarding the vehicle, driver, weather conditions or road network may be included. The information we used as a starting point contained an ID for the vehicle that created the trajectory, a timestamp of the date and time, and the latitude and longitude. The dataset needs to be enriched so that every point p contains the items presented in Table 1. How the dataset can be brought to this form is explained in the subsections below.

Table 1 Input Dataset columns

Vehicle ID
Trajectory ID
Timestamp
Latitude
Longitude
Speed Limit
Length
Node 1
Node 1 Latitude
Node 1 Longitude
Node 2
Node 2 Latitude
Node 2 Longitude

4.1.1 Map Matching

Now imagine partitioning the spatial domain into a graph representing the underlying road network. Roads are represented by the graph's edges and intersections between them are represented by the nodes. Each data point of the trajectory is mapped onto an edge of the graph. Map matching is a whole field of study on its own and there are many algorithms that aim to improve the accuracy of the mapped results by taking into account more data points of the trajectory and parameters such as orientation and velocity[9,10,11,12]. For example, the algorithm developed by Goh et al. in [10] is based on the Hidden Markov Model which sequentially generates candidate paths and evaluates them on the basis of their likelihoods and the surviving path with the highest joint probability is selected as the final solution. Such algorithms tend to be more robust to noise and sparseness when compared to simplistic methods like the one described below.

In this thesis we use a simplistic approach, where each data point is mapped to its nearest edge of the graph based on its coordinates. This is done using OSMnx's get_nearest_edges method which creates equally distanced points along the edges of the network. Then, these points are used in a BallTree search [13] to identify which is nearest to the examined data point and the data point is mapped to the corresponding edge. A BallTree is a binary tree with a hierarchical structure. First, two clusters (hyperspheres) are created. Each point belongs to either the one or

the other sphere based on shortest distance to the sphere's centroid. Each sphere is subdivided into two spheres and that is repeated down to a specified depth. The search exploits this distance metric as for each new data point only the nearest sub-sphere's edge points will need to be searched. In Figure 2 the black rhombuses represent the equidistanced points of the map matching algorithm while the red Xs represent the examined trajectory's data points. In this example, the first three points would have been mapped to the edge between nodes 1 and 2 (edge (1,2)) and the following two would have been mapped onto edge (2,3)

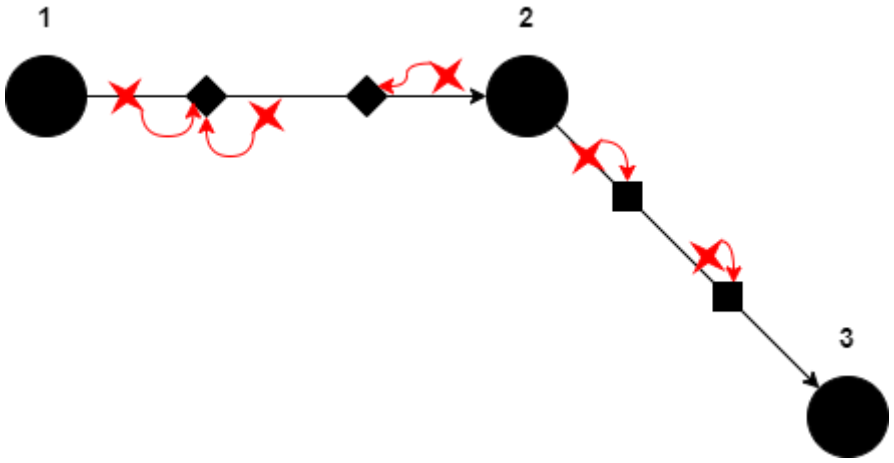


Fig. 2 Map matching points to edges.

The latitude and longitude of the two nodes containing the matched edge are also extracted from OSM and so is the speed limit of the road segment that the edge belongs to. Each data point of the dataset is enriched with this additional road network information.

4.1.2 Trajectory ID

A second preprocessing step required on the input data is defining what is considered to be a unique trajectory and therefore how the dataset will be divided into trajectories with their unique IDs. It is logical that each trajectory can only be created by a single vehicle. The dataset is sorted first by vehicle ID and then by timestamp and examined point by point. A counter is created and if the vehicle ID is not the same as the previous point or if the time delta between them is greater than 5 minutes the trajectory ID counter is incremented by 1. The trajectory ID attached to each data point is a combination of the corresponding values for the vehicle ID and the counter. The 5 minute time delta limit was chosen based on our intuition that the driver is likely to have made a stop or turned off the GPS device if there is such a large gap between the points. The sampling rate for most such GPS devices is in the order of a few seconds so the selected cut-off should not be affected by variations in the sampling rate. If the sampling rate of the examined dataset is in the order of minutes or greater then a different approach would be required.

4.2 Theory and Implementation

Consider the enriched trajectory dataset discussed above. Each point has been mapped to the corresponding edge of the graph. The temporal axis is also divided at a set time interval into a set of equi-sized non-overlapping temporal intervals that define a set of temporal partitions. Therefore, each edge is defined in the spatial domain by the two nodes enclosing it and in the temporal domain by the temporal partition.

Table 2 Description of symbols used

Symbol	Description
D	Dataset
t	A trajectory in the form of a sequence of points
p ∈D	A point of the dataset
G	Graph of the road network
S	3D space partitioning S = {e_i,…,e_n}
e_i ∈G	The i-th edge of G
x_i	The attribute value of edge e_i
$w_{i,j}$	The weight between edge e_i and e_j
a	A parameter used in the weights
n	Number of edges in G
G_i^*	The Getis-Ord statistic for edge e_i

4.2.1 Attribute Value

For each edge e_i an attribute value can be calculated based on the trajectories that have passed through it. For each trajectory t this is defined as a reference speed divided by the average speed of the examined vehicle while traversing the edge. The attribute value for the edge is the average of these values for all trajectories going through it:

$$\bar{X} = \frac{\sum_{t \in e_i} \frac{V_{ref}}{\left(\frac{l}{t_{exit} - t_{in}}\right)}}{m}$$

where V_{ref} is the speed limit of the road the edge is part of, l is the length of the edge, t_{in} and t_{exit} are the timestamps when the trajectory entered and exited the edge respectively and m is the number of trajectories passing through the edge. This definition of the attribute value signifies that the slower a vehicle is moving through an edge, compared to the expected average speed

for the edge, the higher the edge's attribute value will be. The attribute value was defined as such to account for:

- Edges have different lengths which is accounted for by looking at the average speed of the vehicle through the edge.
- The reference speed, in this case the speed limit, varies between edges which is why it is included in the numerator.
- Not all edges have the same number of lanes which is why the average value of the individual trajectories is taken rather than the sum.

The speed limit of the road was chosen as the reference speed because the information is readily available through OSM and it is a good indicator for the expected speed of a vehicle traversing the edge. A better choice for the reference speed of each edge would be the free flow speed of the edge, which is defined as the average speed a vehicle traverses the edge at when there is no congestion. This metric is usually calculated by taking the average speed of the 5-15% fastest trajectories through the edge. However, calculating an accurate estimate for the free flow velocity would require a very large dataset of trajectories, relating to the specific part of the road network under examination, with multiple trajectories passing through each edge.

The length of each edge has been calculated in the map matching step of pre-processing. The length is defined as the geodesic distance between the two nodes linked by the edge and is calculated using the **distance** method of the **geopy** python library. The geodesic distance is the shortest distance on the surface of an ellipsoidal model of the earth.

Entry and exit times for edges are calculated as follows. The points of each trajectory are examined individually and whenever the examined point lies on a different edge from its previous point and the two edges share a node in common a linear interpolation between the coordinates and timestamps of the two points and the coordinates of the node is performed:

$$t_{node} = t_2 - (\frac{d_2}{d_1 + d_2} \times (t_2 - t_1))$$

where t_1,t_2 are the first and second points' timestamps respectively, d_1,d_2 are the geodesic distances between the node and points 1 and 2 respectively and t_{node} is the timestamp at which the trajectory passed the node.

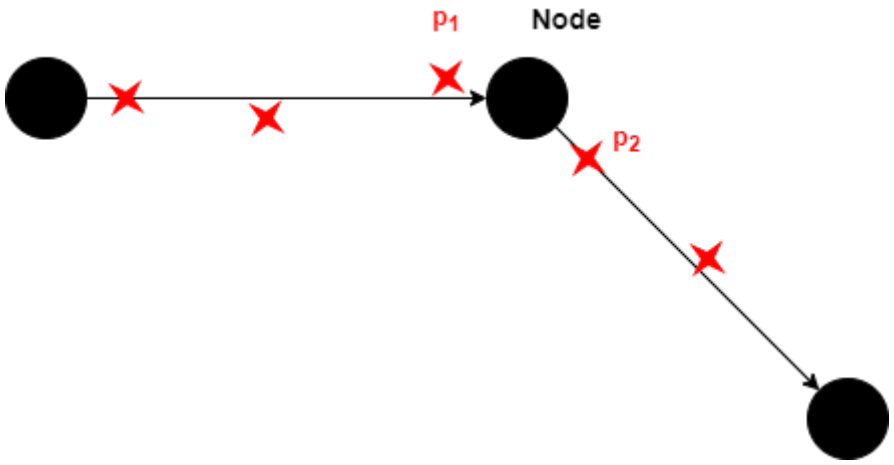


Fig. 3 Calculating the node time.

So in the example in Figure 3 the t_{node} that would be calculated as explained above is the time at which the trajectory exits the first edge (t_{exit}) as well as the time at which it enters the second edge (t_{in}). Once both t_{in} and t_{exit} have been calculated for an edge, the trajectory's attribute value for that edge can be calculated as defined previously. It is important to note that this method does not output the attribute value for edge cases, i.e., the first and last edges the trajectory passes through as the entry and exit time, respectively, would not have been calculated. Also points of the trajectory that have been mapped erroneously, and as a result the two edges do not share a node in common, could lead to some attribute values for these edges of the trajectory to be lost. Discarding edge cases has been deemed acceptable as we work with the assumption of a very large dataset of trajectories and the first and last edge are a very small part of each trajectory. Issues regarding erroneous map matching could also be eliminated through the use of a better map-matching algorithm that uses the entire trajectory to map each point rather than mapping each point individually.

4.2.2 Congestion Hotspot Analysis

This work focuses on the problem of congestion hotspot analysis. It aims to identify statistically significant spatio-temporal areas (edges of the graph) that constitute hotspots. Using the attribute values of all the edges that succeed the examined edge (e_i) and the weights between them and e_i, a G_i^* value can be calculated. The Getis-Ord G_i^* defined in section 3 has been modified to fit our application as follows:

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

where x_j is the attribute value for edge j, $w_{i,j}$ is the spatial weight between edges i and j, z is equal to the total number of edges succeeding e_i, n is equal to the total number of edges, and:

$$\bar{X} = \frac{\sum_{j=1}^{n} x_i}{n}$$

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

Every edge has a weight between it and its succeeding neighbouring edges. The weight is dependent on the network distance between e_i and e_j as the greater the distance between them the smaller e_j's influence on e_i will be. Only succeeding edges will have an effect on the G_i^* value as all proceeding edges will essentially have a weight of 0. This is due to our intuition that edge e_i is likely to experience congestion if the edges in front of it are congested and thus vehicles on e_i have nowhere to go. Therefore the weight between e_i and succeeding edge e_j is defined as:

$$w_{i,j} = a^{1-r}$$

Where a>1 is an application-dependent parameter and r is the network distance between e_i and e_j in number of edges. For direct neighbours where the distance r=1 the weight is 1 and follows an exponential decay for all subsequent neighbours ($w_{ij} = \frac{1}{a}, \frac{1}{a^2}, \frac{1}{a^3}$,etc.). The same applies for temporal distance between edges in number of time steps between them and r is then the maximum value of the network and temporal distances. Weights are calculated for out-edges of both previous and subsequent time steps. In the example in Figure 4, the network distance between e_i and e_j is 2 edges and the temporal distance is 1 time step then $r = \max(2,1) = 2$. In the same examples red edges have a network distance of 1 from e_i and blue edges 2. Note that edge e_i is also coloured red as it will have the same weight, for itself, as that of edges 1 step away. The preceding edge (in-neighbour), for both temporal partitions, is not considered in the calculation of G_i^*, essentially having a weight of 0.

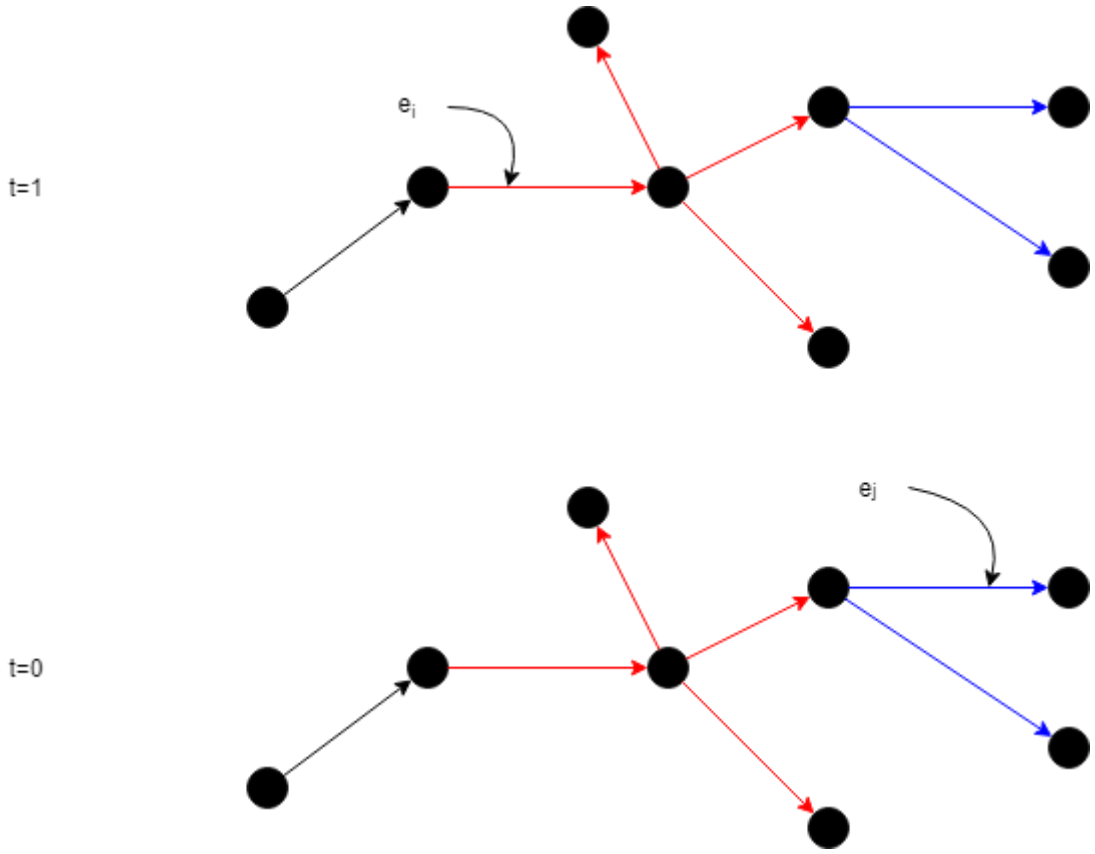


Fig. 4 Calculating distance between e_i and e_j.

The problem that this work focuses on can be formally defined as follows:

Given a data set D of vehicle trajectories on a road network and a graph representation G of said network, find the top-k most statistically significant edges according to the G_i^* statistic, so that: $G_i^* \geq G_j^*$, $\forall e_i \in$ TOPK, $e_j \in$ S−TOPK.

5

Distributed Offline Algorithms

The development of the algorithms that aim to solve this work's problem are discussed in this section. The proposed solution is implemented on Apache Spark and is meant to be run on a computer cluster with multiple nodes. Figure 5 shows the main operations executed by the algorithm.

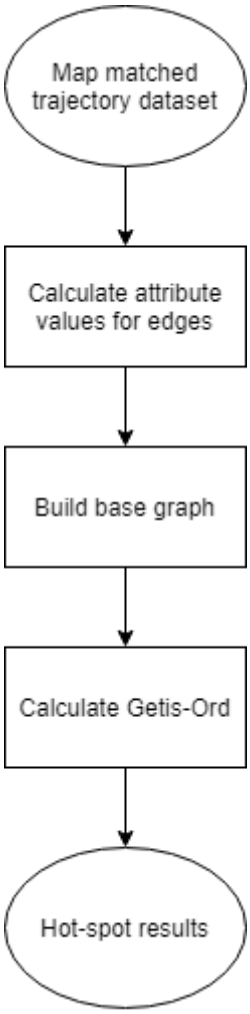


Fig. 5 Flowchart of operations.

5.1 Operations Overview

The solution to the problem has been split into three main steps. The first step is calculating the attribute value for all edges in the spatio-temporal partition of the road network. This is done using the formula introduced in the previous section where, for each edge, the average of the attribute values for all trajectories passing through it is taken. Step 1 is executed once for each data point in the input dataset O(n) and outputs an RDD of attribute values with fewer elements than the input dataset. Then in the second step of the algorithm the mean and standard deviation of all edges' attribute values are calculated. In this step an overall graph, for all timesteps, containing the spatial network relationships of all edges is created. This graph will be used in the Getis-Ord step to calculate the network distance between edges. The mean value and standard deviation are also used in the formula that calculates the G_i^* scores. The performance of step 2 is dependent on the size of the attribute value RDD O(n). The third step calculates the G_i^* statistic (z-scores) for the edges using the Getis-Ord formula and takes as inputs the attribute values calculated previously, their mean and standard deviation, and the overall graph. The exact algorithm is $O(n^2)$ while the approximate algorithm is O(n). The final step of the algorithm would then be to select the top-k edges with the highest z-scores.

5.2 Calculating Attribute Values

Pseudocode 1: Spark operations for calculating the attribute value

```
1  Input: trajectoryRDD, temporal_parameters
2  Output: attRDD
3  function
4  attRDD = trajectoryRDD.groupByKey()\
5     .mapValues(sort_group)\
6     .flatMap(lambda j:calculate_attribute(j,temporal_parameters))\
7     .aggregateByKey(calculate sum and count)\
8     .mapValues(lambda v: v[0]/v[1])
```

The first step in the solution takes as inputs the trajectory dataset as an RDD with the trajectory ID as key and the entire line as value and the temporal partitioning parameters. The time parameters are: the earliest timestamp in the dataset, the timestep chosen and the total number of steps (temporal partitions). The RDD is then grouped by key (line 4), i.e., by trajectory ID, and the groups are sorted based on timestamps (line 5). The groups are then flat mapped (line 6) calling the **calculate_attribute** function (explained below) for each trajectory ID group. The resulting RDD of edge IDs as keys and attribute values as values is aggregated by key (line 7). This calculates for each edge ID the sum of the attribute values for all trajectories going through

it and a counter of these trajectories. The final map operation (line 8) divides the sum by the counter to calculate the average attribute value for each edge. The final output of this step is the attribute RDD containing key value pairs of edge IDs and average attribute values.

5.2.1 Calculate_attribute Function

Pseudocode 2: Calculate_attribute function

```
1  Input: Trajectory point group, Temporal Parameters
2  Output: Tuple(edge ID, att)
3  function
4  for item in group:
5     prev=Point(last_item)
6     curr=Point(item)
7     if not on the same edge:
8        if (curr.Node1== prev.Node1):
9           node_time= calculation of the common node time stamp
10          t_bracket= int((node_time-dataset_min_Time)/timeStep)
11          edge_id_entry=(curr.Node1, curr.Node2)
12          edge_id_exit=(prev.Node2,prev.Node1)
13          entry_dict[edge_id_entry]=node_time
14          if entry_dict.get(edge_id_exit) is not None:
15             att= calculate att value
16             yield (edge_id_exit,t_bracket),att
17       elif (curr.Node1== prev.Node2):
18          equivalent to above with the appropriate edge ID directionality
19       elif (curr.Node2== prev.Node1):
20          equivalent to above with the appropriate edge ID directionality
21       elif (curr.Node2== prev.Node2):
22          equivalent to above with the appropriate edge ID directionality
```

This function takes as input a trajectory ID group of points and the temporal partitioning parameters of the dataset and returns key value pairs of edge IDs as keys and the corresponding attribute values as values. This is done by comparing each data point to the previous point. When the two points are not mapped on the same edge (line 7) but the edges they've been mapped to are direct neighbours (lines 8,17,19,21), i.e., share a node in common, the exit time for the previous edge and the entry time for the current edge are calculated (line 9) as described in section 4.2.1 using the formula $t_{node} = t_2 - (\frac{d_2}{d_1+d_2} \times (t_2 - t_1))$. The node shared by the two edges is used as the entry node for the current edge (line 11) and the exit node for the previous edge (line 12) when constructing the directional edge ID. The calculated entry time for the current edge is then stored in a dictionary (line 13) with entry edge ID as the key and the entry time as the value. The ID for the previous data point's edge (exit edge) is then searched for in the entry edge dictionary (line 14), if an entry time for this edge ID exists in the dictionary

the attribute value is calculated (line 15) as described in section 4.2.1 using the formula $\bar{X} = \frac{\sum_{t \in e_i} \frac{v_{ref}}{l/(t_{exit} - t_{in})}}{m}$. Every time the above statements hold true the function yields a key value pair with the edge ID as key and the attribute value as value (line 16). The edge ID is constructed as ((node1,node2),temporal partition) where nodes 1 and 2 are the two nodes connected by the edge, their order showing the direction (in this case node1 → node2). The temporal partition it belongs in is calculated based on the edge's exit time and falls between 0 and the total number of partitions in the dataset.

5.3 Calculating Broadcast Variables

Pseudocode 3: Calculating broadcast variables

```
1  Inputs:att_list, temporal_parameters
2  Outputs:G,attmean,attstd,n_edges
3  function
4      attmean=np.mean(att_list_values)
5      attstd=np.std(att_list_values)
6      edges=unique edge IDs excluding temporal partition
7      n_edges=len(edges)*temporal_partitions
8      G = nx.DiGraph()
9      G.add_edges_from(edges)
10     G=G.reverse()
11     return sc.broadcast(Parameters(G,attmean,attstd,n_edges))
```

In the second step of the algorithm the attribute RDD created in the previous step is used as an input. This is collected as a list on the central node. The mean and standard deviation of the attribute values are calculated using the numpy library (lines 4,5). A list of all unique spatial edge IDs, i.e., only the IDs of nodes 1 and 2 excluding the temporal partition information, is created (line 6). The total number of edges in the 3D space partitioning S is calculated by multiplying the length of this edge list by the number of temporal partitions (line 7). Then a networkX directed graph is created, the edges in the edge list are added to it and the direction of the graph is reversed. The reason why the graph is reversed will be explained in the following section. The output of this step is a broadcast object containing the reversed graph, the attribute mean, the attribute standard deviation and the total number of edges in the 3D space partitioning.

5.4 Calculating z-scores Using the Getis-Ord Statistic

Pseudocode 4: Spark operations for calculating z-scores

```
1 Inputs: attributeRDD, temporal_parameters, broadcast_variables
2 Outputs: RDD of edge IDs and z-scores
3 function
  resultRDD=attRDD.flatMap(lambda j: GetisOrdCalc(j, broadcast_variables,
4 temporal_parameters))\
5    .reduceByKey(lambda x,y:(x[0]+y[0],x[1]+y[1],x[2]+y[2]))\
6    .mapValues(lambda j: Gi*_formula(j,params))
```

The final step of the algorithm is calculating the z-scores using the Getis-Ord G_i^* statistic. The inputs to this step are the attribute RDD, the temporal parameters and the broadcast variables from the previous step. The first step flat maps the attribute RDD (line 4) calling the GetisOrdCalc function (section 5.4.1) for each line of the RDD. The outputs of this function is an RDD with edge IDs as keys and 3-tuples of the weighted attribute value, the weight and the squared weight as values. The second step (line 5) reduces the previous RDD by key calculating, for each edge ID, the sums for the weighted attribute value, the weight and the squared weight. The final mapValues operation (line 6) takes these sums and applies the G_i^* formula $G_i^* = \frac{\sum_{j=1}^{z} w_{i,j}x_j - \bar{X}\sum_{j=1}^{z} w_{i,j}}{S\sqrt{\frac{[n\sum_{j=1}^{n} w_{i,j}^2 - (\sum_{j=1}^{n} w_{i,j})^2]}{n-1}}}$. The result of this operation is an RDD consisting of edge IDs and their Getis-Ord z-scores.

5.4.1 GetisOrdCalc Function

Pseudocode 5: Calculating Getis-Ord weighted attributes, weights and squared weights

```
1  Inputs:attRDD line, temporal_parameters, broadcast_variables
2  Outputs: Tuple(edge ID, Tuple(attweight,weight,sqweight))
3  function
4  key,att=line
5  nodes,t_bracket=key
6  ui,vi=nodes
7  t_min=0
8  t_max=times.n_Steps
9  nn = nx.ego_graph(G,ui)
10 for x in range(t_min,t_max):
11    if abs(t_bracket-x)==0:
12       weight=params.a**(0)
13    else:
14       weight=params.a**(1-abs(t_bracket-x))
15    sqweight=weight**2
16    attweight=(att*weight)
17    yield ((ui,vi),x),(attweight,weight,sqweight)
18    if n>=1:
19       for (uj,vj) in nn.edges:
20          weight=params.a**(1-max(nn.degree(vj),abs(t_bracket-x)))
21          sqweight=weight**2
22          attweight=(att*weight)
23          yield ((vj,uj),x),(attweight,weight,sqweight)
```

This function takes as inputs a line of the attribute RDD, the temporal parameters and the broadcast variables. First, the edge's node IDs and attribute value are extracted from the line (lines 4-6). Variables t_min and t_max are also defined (lines 7,8) which are the first and last temporal partitions. Then the networkX method **ego_graph** is used (line 9) on the entry node of the edge (ui) with the reversed graph from the broadcast variables. This produces a graph (nn) containing all the predecessors of the examined edge and their network distances from it. In the next step (lines 10-17) the algorithm loops over all temporal partitions and yields the examined edge's ID and its weighted attribute, weight and squared weight. The weight used in calculating these is based on the temporal distance (number of partitions) between the edges. In the case where this distance is zero the weight is set to 1, which is the same as the weight for an edge with distance 1. In the final step (lines 18-23) if the edge has predecessors (in-neighbours), for each neighbour, the weighted attribute, weight and squared weight are calculated. Then, the neighbour's edge ID, with the nodes in reverse order to account for graph G having been reversed previously, the weighted attribute, weight and squared weight are yielded.

5.5 *Approximate Algorithm*

Pseudocode 6: Approximate solution modifications

```
1  Inputs:attRDD line, temporal_parameters, broadcast_variables
2  Outputs: Tuple(edge ID, Tuple(attweight,weight,sqweight))
3  function
4  key,att=line
5  nodes,t_bracket=key
6  ui,vi=nodes
7  if t_bracket-r>=0:
8      t_min=t_bracket-params.radius
9  else:
10     t_min=0
11 if t_bracket+r+1<=timesteps:
12     t_max=t_bracket+r+1
13 else:
14     t_max=timesteps
15 nn = nx.ego_graph(G,ui,radius=r)
```

The previously described algorithm is exact and uses all succeeding neighbour edges from all the temporal partitions in calculating the solution. As a result, its computational cost is very high and the larger the graph G examined or the finer the temporal partition timestep, i.e., the greater the number of temporal partitions, the higher its cost will be. Since each edge's attribute value needs to be transmitted to all other succeeding edges of the graph and for all temporal partitions, the computational and network costs are $O(n^2)$. This type of cost can become prohibitive very fast as n grows.

In order to combat this an approximate algorithm has been developed. This algorithm will compute an approximation of the G_i^* value by only considering neighbours at a maximum distance of r, whether that is network distance or temporal distance. Network distance is measured in number of edges while temporal distance is measured in number of partitions (timesteps). The only change to the code of the exact algorithm that was made can be seen in pseudocode 6. The GetisOrdCalc function has been modified so that t_min and t_max are defined within the distance r. The networkX method **ego_graph** has also been modified by including the radius parameter equal to r. The produced graph (nn) will then only contain the predecessors of the examined edge within a distance of r.

Based on the selected exponential decay formula of weight with distance, the further away an edge is from e_i the smaller the effect it will have on the G_i^* value will be. Accordingly, far away edges will have a negligible effect on the G_i^* value and can be ignored without affecting accuracy significantly. In the example presented in Figure 6, for r=2 only the coloured edges

will be taken into consideration. As with the exact algorithm, network in-neighbours are ignored but edges that are at a distance greater than 2 are also ignored, essentially setting their weight to 0. Edges coloured red are at a distance of 1 from e_i and blue coloured edges are at a distance of 2. As already stated, this applies to both network and temporal distance and the edge distance is defined as the maximum value of the two. Specifically examining the temporal partition t=1 (or t=5) we can see that even edges at a network distance of 1 are coloured blue as the temporal distance of 2 is used (max(1,2)=2). Finally, as can be seen from the figure, temporally, both previous and following neighbours are considered.

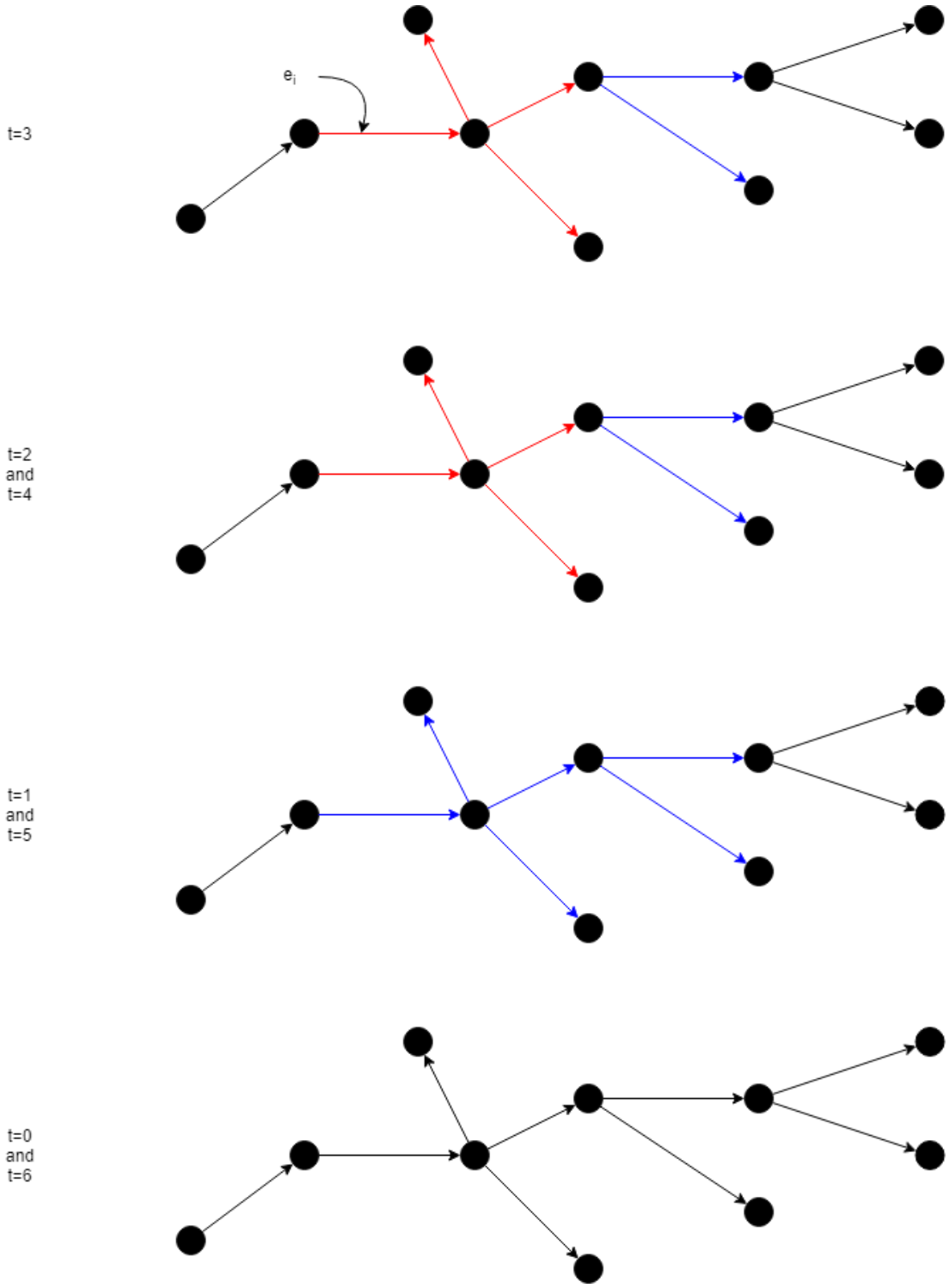


Fig. 6 Distance cut-off example.

6

Experimental Evaluation

The performance of the solutions to the problem is evaluated in this section. The algorithms were implemented in Python 3.7 using Apache PySpark version 2.4.3. The experiments were conducted on a single node with 4 CPU cores and 8GB of RAM

6.1 Evaluation Dataset

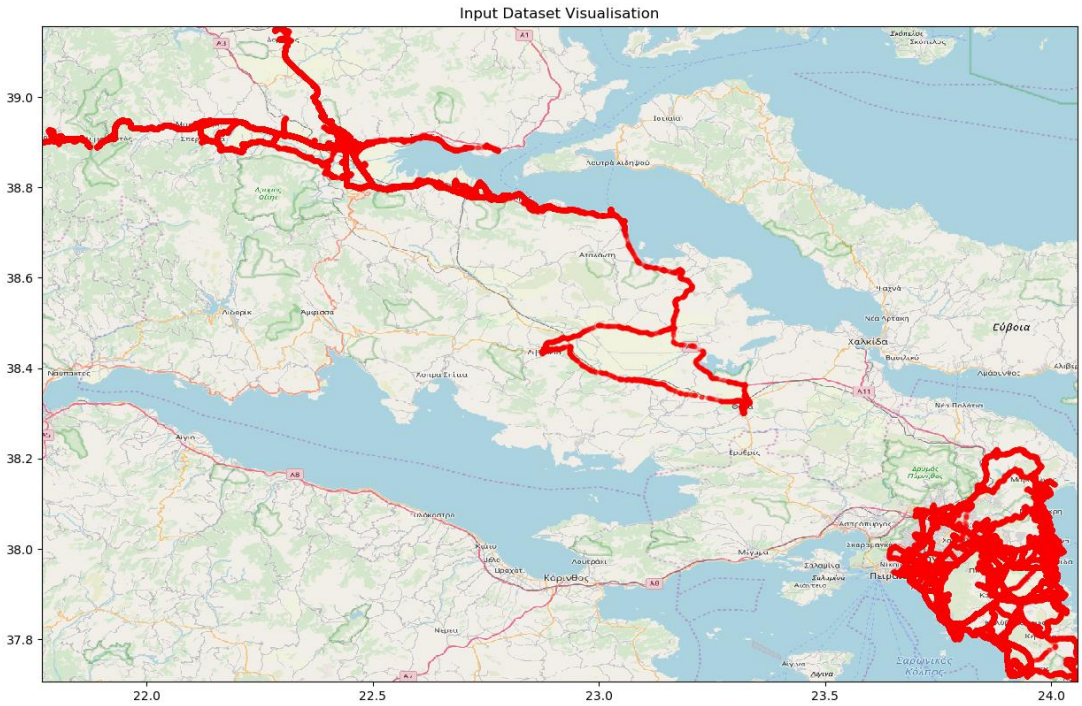


Fig. 7 Input Dataset Visualisation

The dataset used for the evaluation consists of real GPS data. The data describe the trajectories of three vehicles in Greece for the time period between 02/07/2018 and 30/11/2018. The data points are visualised in red in Figure 7. The dataset is 262 MB in size and contains a total of

1,691,004 data points that have been divided into 1,285 trajectories following the guidelines set in section 4.1.2. The dataset is sparsely distributed on the temporal axis, averaging only about 9 trajectories per day. It is our intuition, based on how traffic tends to build up in cities, that the timeframe a few hours before and after the examined time partition will be relevant to how much congestion there will be, i.e., if an edge is congested at 8am it is likely to be congested at 9am. Conversely, if an edge was congested or not 3 days ago is not a good indication on whether it will be congested now. Therefore, the date part of the datetime will be ignored to simulate all 1,285 trajectories occurring on the same day. The dataset is stored as a CSV text file and each line represents a point in the trajectory of one of the vehicles. Each record consists of: {vehicle ID, trajectory ID, datetime, latitude, longitude, speed limit, length of the edge, node 1 ID, node 1 latitude, node 1 longitude, node 2 ID, node 2 latitude, node 2 longitude}.

6.2 Outputs

The outputs of the algorithms will be the top-k edges in the 3D space partitioning with the highest attribute values. This is done by adding an extra step to the algorithms analysed in the previous section. The Spark RDD function rdd.top(k, key=lambda x: x[1]) is used which returns the top-k elements of the specified RDD.

6.3 Evaluation Methodology

The algorithms will be evaluated based on execution time. The time taken to execute the algorithm steps, excluding any overhead due to Spark setting up, is measured. Each step is timed individually in number of seconds. Additionally, the overall time taken has been measured for subsets of the dataset ranging from 10,000 data points to the full dataset to investigate how the algorithms scale with the size of the dataset. The total number of times attribute values were calculated from the original dataset will also be compared to the total number of times trajectories changed edges. The total number of edges for which an attribute value was calculated will also be calculated and compared to the total number of unique edges in the original dataset. Experiments will be conducted by varying the following parameters: (1) the size of the temporal partitions and (2) the distance r within which neighbouring edges will be considered.

Table 3 Parameters used for evaluation (Bold are default values)

Parameter	Values
Temporal partition size T_s (hours)	0.25, 0.5, **1**, 24
r	2, **5**, 10, All (Exact)

The steps of the algorithm that will be timed are the following:

1. Attribute values are calculated for the edges of the 3D spatio-temporal partition S.
2. The mean value and standard deviation of the attribute values of the edges are then calculated and a graph of the edges is created. These are stored as broadcast variables.
3. The Getis-Ord z-scores are calculated for the edges based on the attribute value RDD and the broadcast variables.
4. The top-k edges with the highest z-scores are returned.

6.4 Results

6.4.1 Attribute Values Captured

As already mentioned, the first step of the algorithm calculates entry and exit times, upon which the calculation of the attribute values is dependent, whenever a trajectory moves from one edge to a neighbouring edge. Edge cases, i.e., the first and last edge of each trajectory, are discarded and so are problematic edges where two consecutive points have been mapped to non-consecutive edges. Thus, an important metric is how many edges of the original map matched dataset are lost and, since edge cases constitute only a small percentage of each trajectory, this will be a good indication of the number of problematic edges. In the original map matched dataset there are 32,988 unique edges while an attribute value is calculated for 26,404 unique edges, which is equivalent to an edge capture rate of 80%. However, a more important metric is for what percentage of edge changes an attribute value was calculated. There are 417,790 edge changes in the dataset while only 201,335 attribute values were calculated, a capture rate of 48%. Since there are 2 edge cases per trajectory there is a total of approximately 2,500 edge cases (0.6% of total cases) in the entire dataset. Almost the entirety of lost edge changes is due to problematic cases. Therefore, the accuracy of the map matching algorithm used in preparing the input dataset, and its ability to deal with problematic data points, is critical to the efficiency of the algorithm.

6.4.2 Varying the Size of the Dataset

In table 4 and Figure 8, the results for variations to the size of the input dataset are presented. The algorithm was run for sections of the full dataset with 10,000 points, 100,000 points, 1,000,000 points and the full dataset of 1,691,004 points. Examining the results, we can see that the algorithm's execution time scales linearly O(n). Execution time is slightly higher than

expected for small datasets, likely due to Spark's initialisation overheads having a greater impact in the total execution time of small datasets. For even larger datasets we would expect the execution time to continue to scale linearly with dataset size. For these results, the approximate algorithm was run with the parameters set to their default values r=5 and T_s=1.

Table 4 Execution Time vs Dataset Size

No. of points	Time (s)				
	Step 1	Step 2	Step 3	Step 4	Total
10,000	1.75	2.90	2.42	1.06	8.14
100,000	3.37	7.11	5.28	1.28	17.04
1,000,000	21.33	36.12	29.49	3.77	90.70
1,691,004	21.72	58.69	41.43	5.49	127.32

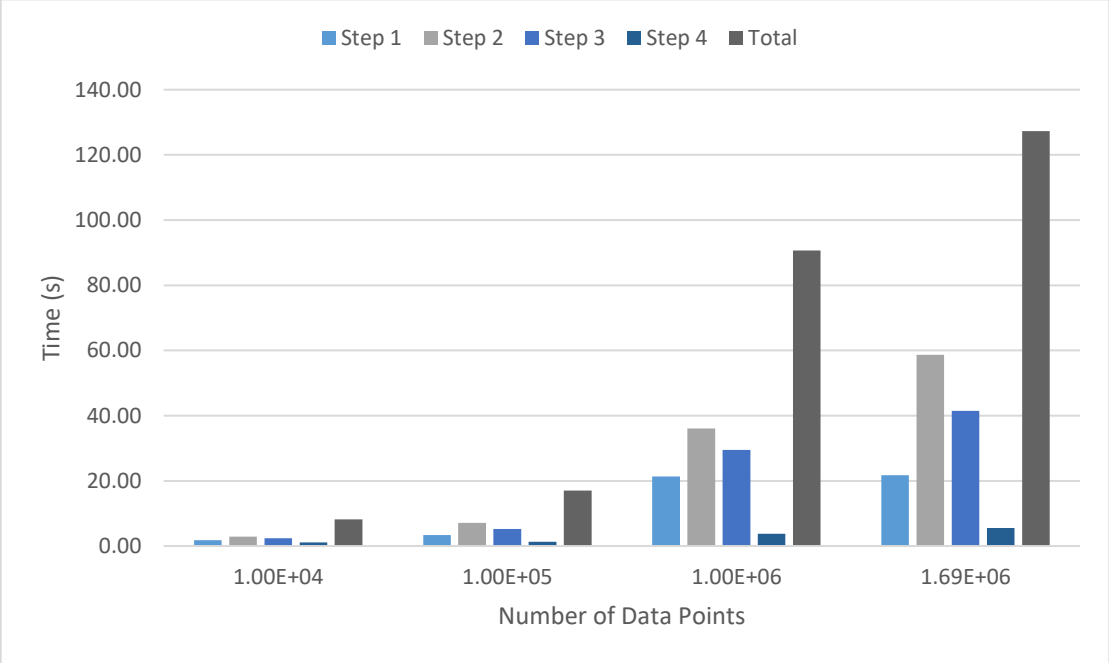


Fig. 8 Execution Time vs Dataset Size

6.4.3 Varying r

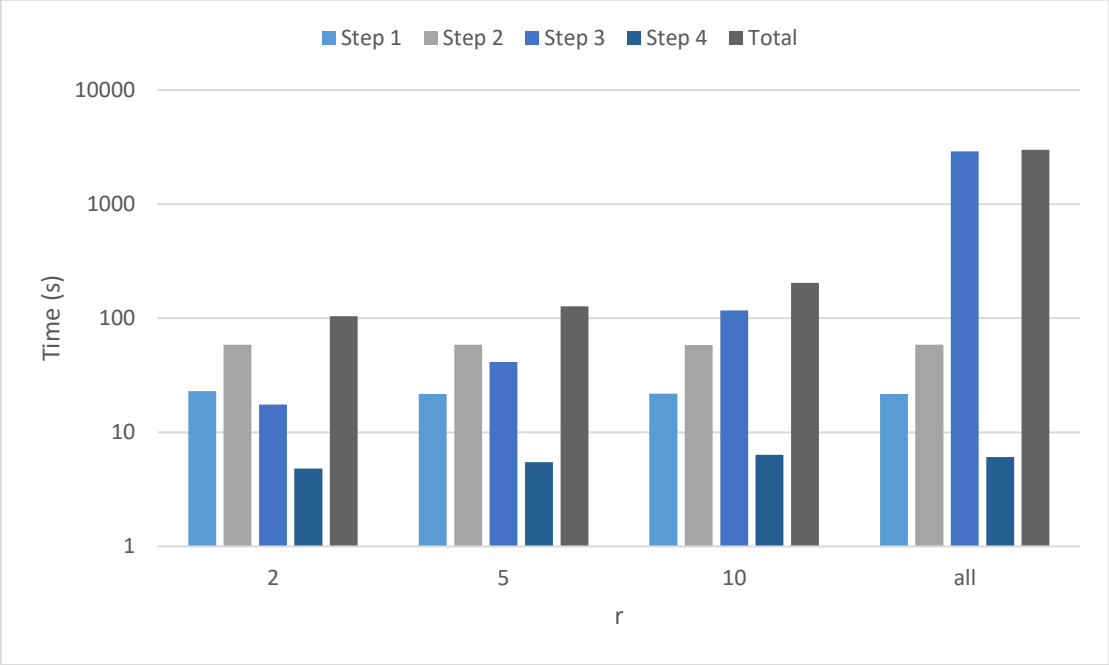


Fig. 9 Execution Time vs r

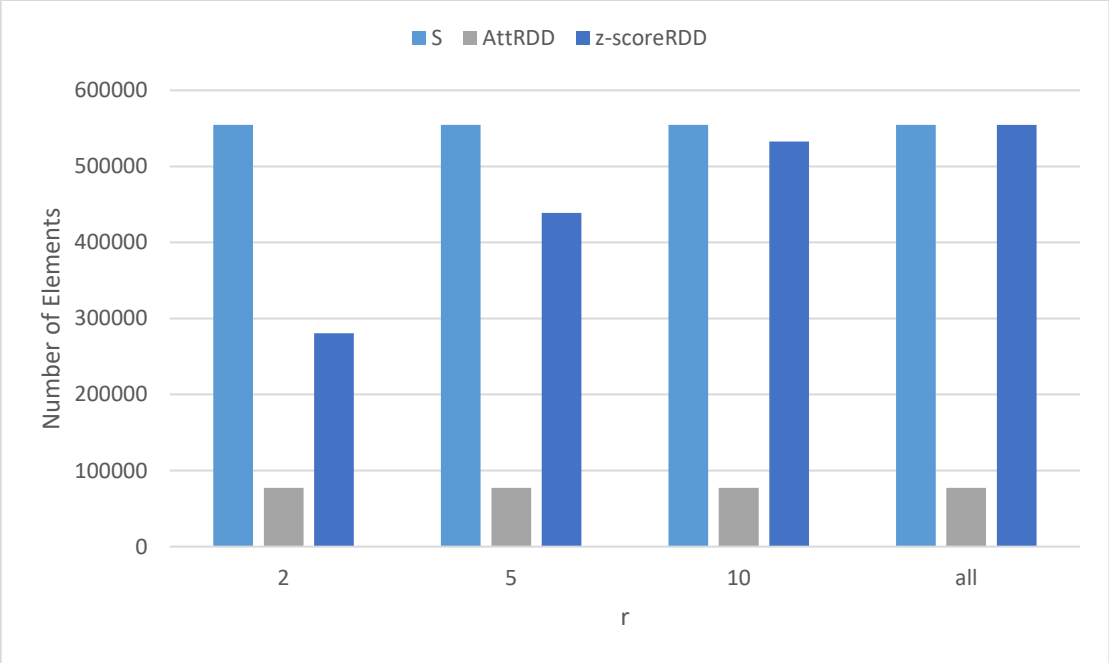


Fig. 10 Number of Edges vs r

In Figure 9 variations in r and the resulting execution times are shown. It is important to note that the time axis has a logarithmic scale with base 10. Examining Figure 9 shows that the total execution time is highly dependent on the value of r selected and the lower the value of r the lower the total execution time. The exact algorithm, that takes into account all out-neighbours for each edge, requires over 28x longer to complete compared to the approximate algorithm

with an r value of 2. Also note that the execution times of steps 1, 2 and 4 are unaffected by variations in r and only the execution time of step 3 is affected. This is expected as the exchange of information between neighbours and all related calculations happen in step 3. The greater the value of r the more the neighbours to which each edge's attribute value needs to be transmitted to is and the more calculations need to be performed to calculate z-scores. This is consistent with Figure 10 where the size of the attribute value RDD (attRDD) is unaffected by variations in r while the z-score RDD grows in size as r increases. This is due to "empty" edges of the attRDD acquiring z-scores based on non-empty neighbours and as r increases so does the likelihood of finding a non-empty neighbour with the exact algorithm producing a z-score for all edges. As steps 1 and 2 are operations on the attRDD, whose size is unaffected by r, it is sensible that their execution time remains constant. Step 3 is the operation that creates the z-score RDD and as the z-score RDD increases in size so does the execution time of this step.

6.4.4 Varying the Temporal Step Size

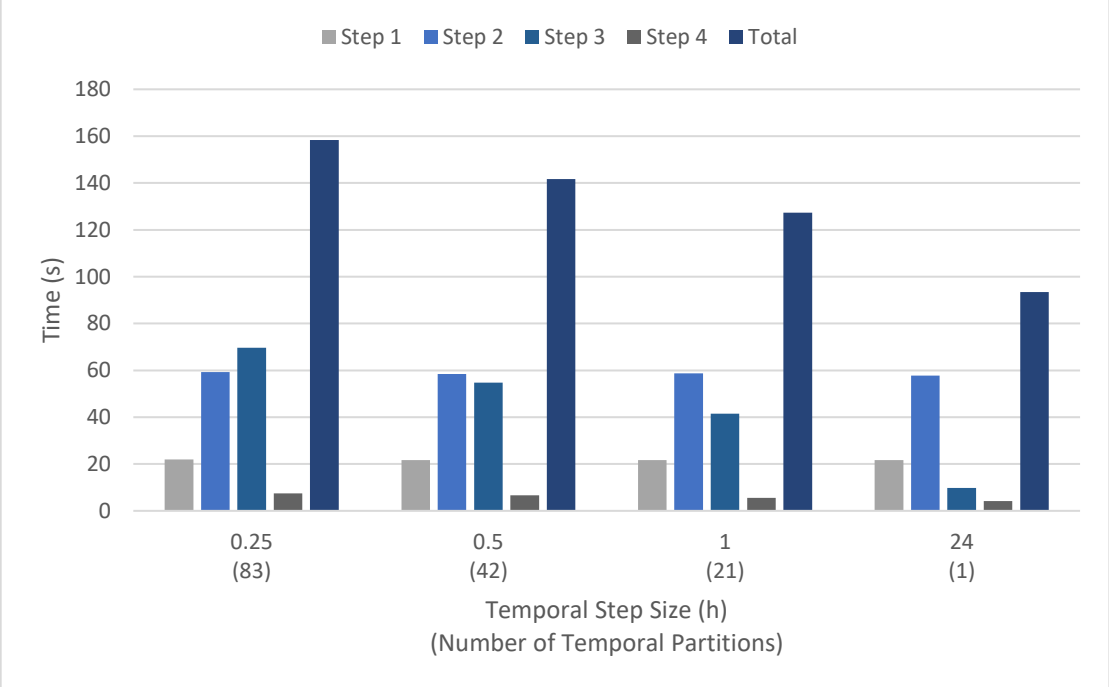


Fig. 11 Execution Time vs Temporal Step Size Ts

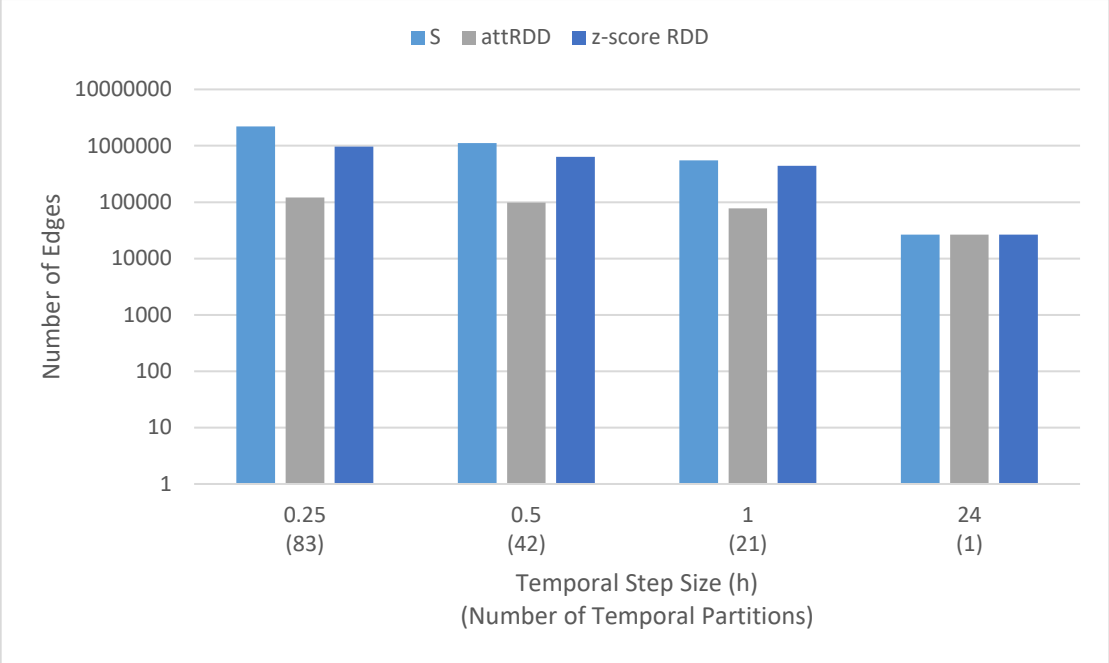


Fig. 12 Number of Edges vs Temporal Step Size T_s

In Figure 11 variations in the temporal step size T_s and the resulting execution times are shown. Examining the figure shows that the total execution is dependent on the temporal step size and the smaller the step, i.e., the more steps there are, the longer the execution time. Step 1 is unaffected by the temporal step size as its execution time is only dependent on the number of points in the input dataset. The network cost of step 2 is dependent on the number of edges in the attRDD, however the execution time remains almost constant with variations in T_s. Looking at Figure 12 we observe that the size of the attRDD does increase as T_s gets smaller but with the exception of T_s=24 hours, where there is only a single temporal partition, the changes in its size are small when compared to the variation of the total number of edges in S and in the z-score RDD. The execution time of step 3 shows the strongest dependence to T_s, as it appears to scale linearly with the size of the attRDD. Finally, the execution time of step 4 also shows a weak dependence to T_s. The variations in the total execution time are again primarily due to step 3 and to a lesser extent to step 4.

6.4.5 Hotspot Visualisation

Figures 14 and 15 help visualise discovered hotspots on the road network. Figures 14 and 15 (a) have been produced using the approximate algorithm with an r value of 5 and a temporal step size of 24 hours, i.e., for a single temporal partition. Figure 15 (b) has been produced using the approximate algorithm with an r value of 10 and the same temporal step size of 24 hours. Figure 15 (c) has been produced using the exact algorithm with the same temporal step size of 24 hours. White edges are edges for which there is no z-score information. For the remaining edges the higher their z-score the darker their colour. Examining Figure 14 we can see that,

while there is a large cluster of hotspots along the national highway, most hotspots are located along the major arteries of cities. This follows our intuition as the road networks of cities tend to experience congestion a lot more often than highways between cities do. Figure 15 is centered around the region of Attica; we can perform a visual comparison between the outputs of our algorithms (a,b and c) and typical traffic for the same region from google maps (d). These results coincide with our expectations and typical traffic patterns as it is observed that a lot of hotspots cluster along major roadways that get highly congested during rushhour in and around the centre of Athens.

Both the approximate and exact algorithms produce a similar hotspot distrinution. However, the approximate algorithm does not capture all the hotspots found by the exact algorithm while other edges have been given a larger z-score than that of the exact algorithm. The attribute values produced by both algorithms range between 0.04 and 5134 however the majority of attribute values have a value lower than 10. Figure 13 shows a histogram of the attribute values capped at 100. The formula for the weight between edges is given by $w_{i,j} = a^{1-r}$. The values of a and r used in calculating the approximate solution (a=2 and r =5) mean that the smallest weight for the furthest edges considered is 1/16. This is only about an order of magnitude smaller than the weight of direct neighbours and since most attribute values are distributed close to zero the effect of edges outside of the range examined by the approximate algorithm, but with abnormally high attribute values (e.g. 100+) should still be significant to the examined edge's z-score. Increaing the values of a and/or r so that the minimum weight considered is 2 or 3 orders of magnitude smaller than that of direct neighbours should bring the results of the approximate algorithm closer to those of the exact. This is seen in figure 14 (b) where an r value of 10 was used bringing the the smallest weight for the furthest edges considered to 1/512.

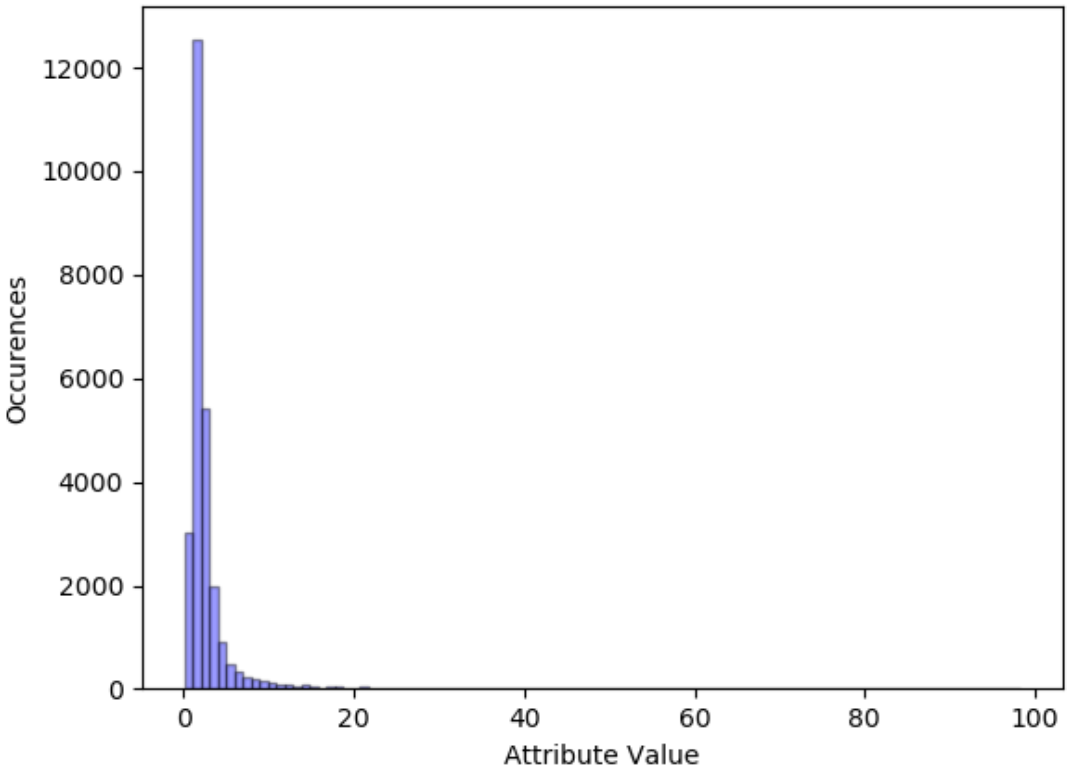


Fig. 13 Histogram of Attribute Values between 0 and 100

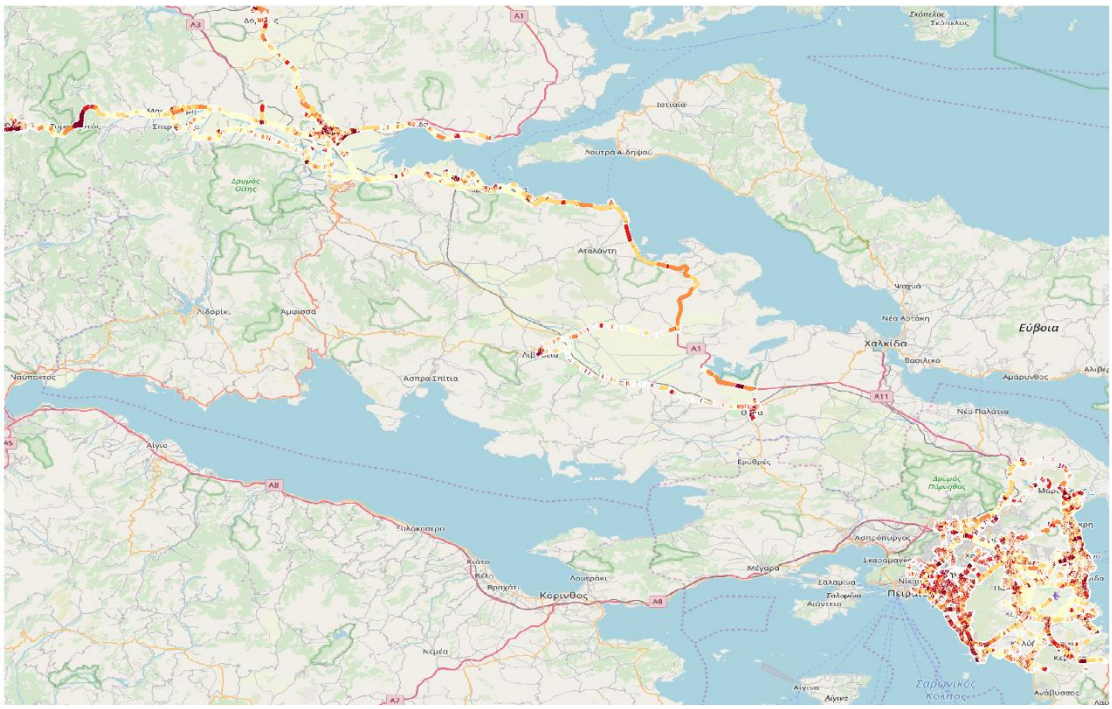
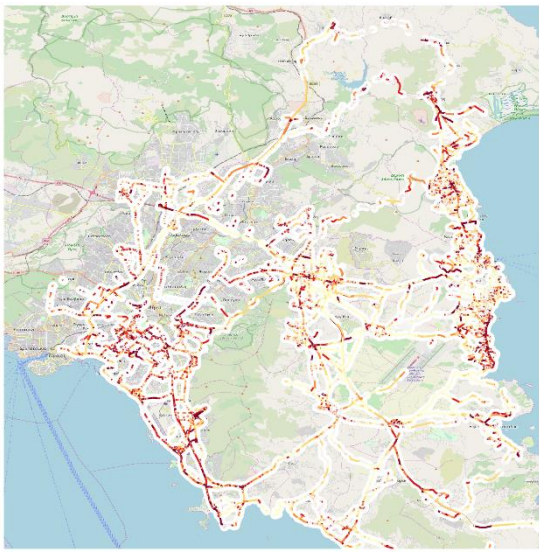
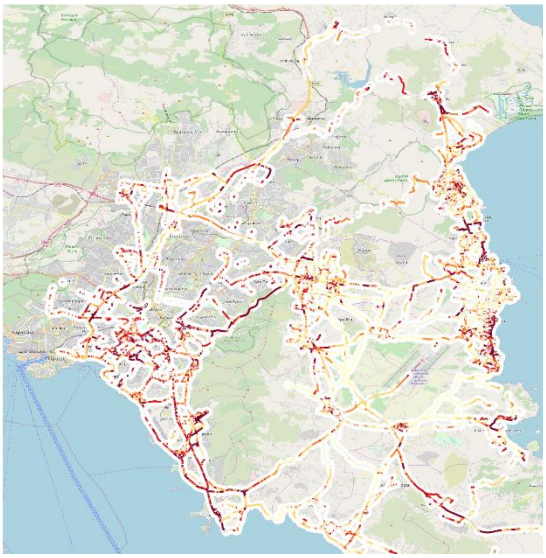


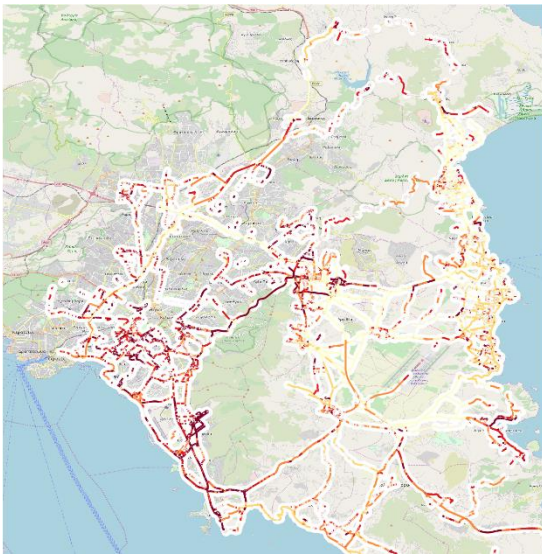
Fig. 14 Hotspot Visualisation for the Entire Dataset



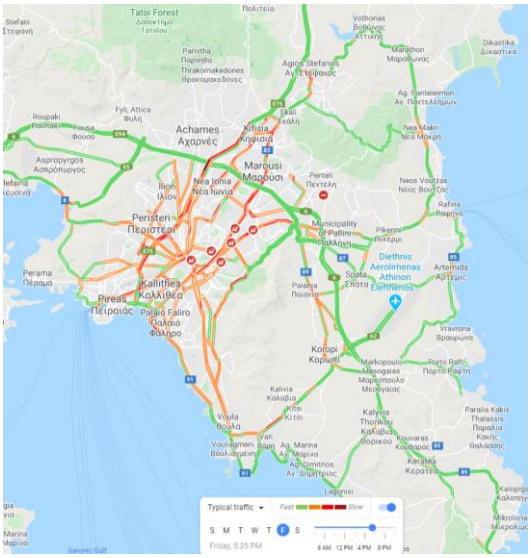
(a) Approximate Algorithm with r=5 Output



(a) Approximate Algorithm with r=10 Output



(c) Exact Algorithm Output



(d) Typical Traffic from Google Maps

Fig. 15 Hotspot Visualisation in Attica

7

Conclusion

In this work, an approach for finding trajectory hotspots on road networks is proposed. This can be useful for many real-life applications in analysing the performance of a road network and discovering underlying issues. Two solutions are proposed, an algorithm that provides an exact solution to the problem and an approximate algorithm that only considers neighbouring edges within a certain distance in calculating z-scores. The approximate algorithm provides significant time savings over the computationally expensive exact solution by sacrificing some accuracy. The efficiency of the algorithm was found to be affected by the accuracy of the map matching algorithm used for pre-processing the dataset. Both implementations utilise Apache Spark and were tested on a dataset consisting of 1,285 trajectories on the road network of Greece. One great advantage of the Apache Spark implementation is that the size of the computing-cluster can be adjusted according to the size of the input dataset to maintain an acceptable execution time for the algorithm.

7.1 Future Work

In this chapter aspects of the solution worth investigating further in future work are discussed. Firstly, trajectory specific map matching algorithms for producing the input dataset, and the effect they have on improving the efficiency of the algorithm, are worth investigating. For the same goal of improving efficiency an implementation of the first step that calculates attribute values for problematic edge cases where, due to a sparser sampling rate, sequential data points have been mapped to edges with a distance greater than one is worth examining. For the approximate solution, the network cost of the second step, where the attribute value RDD is collected to produce the graph and the entire graph is then broadcast to all nodes could possibly be reduced by developing a solution that transmits only the relevant neighbour information for each edge.

The fine tuning of parameters is also worth investigating. Firstly, for large enough datasets, it is worthwhile to examine the effect of replacing the speed limit with the free flow speed of the edge as the reference velocity in calculating the attribute values. Also, by changing the attribute value metric altogether, different aspects of the road network can be examined. For example, the average number of times vehicles came to a stop on each edge can be used as an attribute value metric and allow us to draw different conclusions about the performance of the road network.

8

References

[1] P. Nikitopoulos, A. Paraskevopoulos, C. Doulkeridis, N. Pelekis and Y. Theodoridis, "Hot Spot Analysis over Big Trajectory Data," 2018 IEEE International Conference on Big Data (Big Data), Seattle, WA, USA, 2018, pp. 761-770.

[2] 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.

[3] Yiqun Xie and Shashi Shekhar. 2019. Significant DBSCAN towards Statistically Robust Clustering. In Proceedings of the 16th International Symposium on Spatial and Temporal Databases (SSTD ’19). Association for Computing Machinery, New York, NY, USA, 31–40. DOI:https://doi.org/10.1145/3340964.3340968

[4] X. Tang, E. Eftelioglu, D. Oliver and S. Shekhar, "Significant Linear Hotspot Discovery," in IEEE Transactions on Big Data, vol. 3, no. 2, pp. 140-153, 1 June 2017.

[5] Nikolaos Zygouras and Dimitrios Gunopulos. 2017. Discovering Corridors From GPS Trajectories. In Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (SIGSPATIAL ’17). Association for Computing Machinery, New York, NY, USA, Article 61, 1–4. DOI:https://doi.org/10.1145/3139958.3139994

[6] Benjamin Krogh, Ove Andersen, and Kristian Torp. 2012. Trajectories for novel and detailed traffic information. In Proceedings of the 3rd ACM SIGSPATIAL International Workshop on GeoStreaming (IWGS ’12).

Association for Computing Machinery, New York, NY, USA, 32–39. DOI:https://doi.org/10.1145/2442968.2442973

[7] Kitazato, Tomoya & Hoshino, Miku & Ito, Masaki & Sezaki, Kaoru. (2018). Detection of Pedestrian Flow Using Mobile Devices for Evacuation Guiding in Disaster. Journal of Disaster Research. 13. 303-312. 10.20965/jdr.2018.p0303.

[8] Häsner, M., Junghans, C., Sengstock, C. & Gertz, M., (2011). Online hot spot prediction in road networks. In: Härder, T., Lehner, W., Mitschang, B., Schöning, H. & Schwarz, H. (Hrsg.), Datenbanksysteme für Business, Technologie und Web (BTW). Bonn: Gesellschaft für Informatik e.V.. (S. 187-206).

[9] Quddus, M. A., et.al, A general map matching algorithmfor transport telematics applications.GPS solutions ,Issue 7(3), pp. 157-167, 2003.

[10] Goh. C. Y.et.al, Online map-matching based on hiddenmarkov model for real-time traffic sensing applications, In Intelligent Transportation Systems (ITSC) on 15thInternational IEE Conference, pp. 776-781,2012

[11] He, Z. C, et.al, On-line map-matching framework forfloating car data with low sampling rate in urban roadnetworks , IET Intelligent Transport Systems, 7(4), 404-414, 2013.

[12] Yin, H., & Wolfson., O, A weight-based map matchingmethod in moving objects databases , In Scientific andStatistical Database Management Proceedings on16thInternational Conference, pp. 437-438, 2004.

[13] Omohundro, Stephen M.. “Five Balltree Construction Algorithms.” (2009).